PROGRAMMING THE MACINTOSH® WITH

FUTUREBASIC™

Programming the Macintosh®

with FutureBASIC™

A self-teaching text for first-time programmers

© copyright 1994, Zedcor, Inc., All rights reserved.

By Michael A. Gariepy

Zedcor, Inc.
4500 E. Speedway Blvd, Suite 22
Tucson, Arizona 85712-5305

Business offices: 602.881.8101
Technical support: 602.795.3996 Monday-Friday, 9-5 MST

2nd printing, March 1, 1994

FutureBASIC is a trademark of Zedcor, Inc. All other products and logos mentioned in this document are trademarks of their respective owners.

About this book

The goal of this book is simple: teach you the fundamentals of programming on the Macintosh using *FutureBASIC*. It uses a self-teaching approach so you can learn at your own pace.

This book assumes:

- You have an Apple Macintosh with Zedcor's *FutureBASIC* installed. It makes numerous references to the *FutureBASIC Reference* manual and *Handbook*, so have these books accessible.
- You are familiar with standard Macintosh operations, such as entering and editing text, using the mouse, closing windows, finding files in folders, and so on. If you've never used a Macintosh before and feel uncomfortable with these procedures, please consult the user manuals that came with your Macintosh.

After completing a lesson, compare your answers to those in the back of the book. Your programs don't have to match the answers exactly, as long as they accomplish what was requested and match window output. I strongly encourage you to experiment.

Take your time going through the lessons. Doing a few lessons a day will have far better results than trying to learn everything overnight. The physical act of entering, editing and observing the results is the best way to acquire programming skills.

If you want details about a particular subject, consult the *Reference* manual or *Handbook*. If you get frustrated and need an expert to talk to, call Zedcor technical support at 602-795-3996 (during normal business hours).

When you complete the lessons, you'll learn that computers are not incomprehensible devices, but tools you can use to accomplish an infinite number of amazing things.

Michael A. Gariepy
Tucson, Arizona
March 1, 1994

P.S. I'd like to thank everyone who helped me fine-tune this book. The forty *FutureBASIC* beginner beta sites added greatly to the quality of this document. As usual, John Richetta, one our programming engineers, provided no-nonsense input that smoothed over rough edges. Thanks to all!

The most valuable assistance came from my wife Marcella, a non-programmer (at least before she went through this book). She painstakingly entered each exercise and programming problem, and asked me hard questions about unclear exercises or poorly presented concepts. Her questions often forced me to rethink, and subsequently rewrite, whole sections of the book. Thank you Marcella.

Table of Contents

About this book 4

Set Preferences *before* doing lessons 7
Steps for doing Exercises 8
Steps for doing Programming problems 9

Lesson 0: Getting Started 11
Lesson 1: Printing to a window 15
Lesson 2: Math operations 21
Lesson 3: Text characteristics 27
Lesson 4: Introduction to graphics 33
Lesson 5: Variables for holding numbers 39
Lesson 6: Variable for holding text characters 45
Lesson 7: Reading data from the keyboard 51
Lesson 8: Comparing values with IF 53
Lesson 9: Repeating program lines (looping) 57
Lesson 10: Numeric functions 63
Lesson 11: Counter loop 71
Lesson 12: Functions 75
Lesson 13: Text functions 81
Lesson 14: Reading data in memory 87
Lesson 15: Files 91
Lesson 16: Array variables 97
Lesson 17: Selecting one value out of a list 105
Lesson 18: ResEdit and resources 111
Lesson 19: Events, buttons and menus 117
Lesson 20: Using the mouse 125
Lesson 21: More about windows 129
Lesson 22: Sending output to a printer 133
Lesson 23: Text edit fields 137

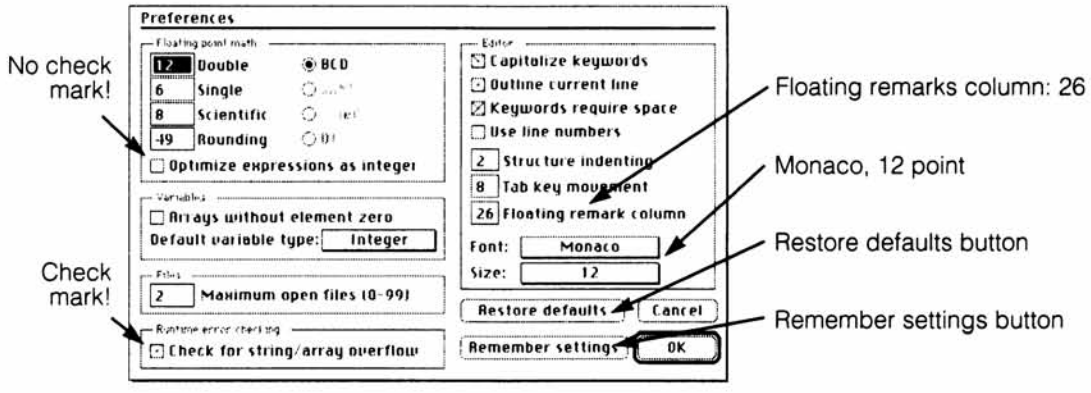
Parting notes 143

Answers 145

Set Preferences *before* doing lessons

To avoid unnecessary frustration when using this book, set-up *FutureBASIC* by following the steps below:

1. Choose *Preferences...* under the Edit menu.
2. Click the *Restore defaults* button.
3. Set *Floating remarks column* to *26*.
4. Set *Font* to Monaco and *Size* to 12.
5. Make sure *Optimize expressions as integer* is NOT checked. The *Preference* window should look like this when you are finished:



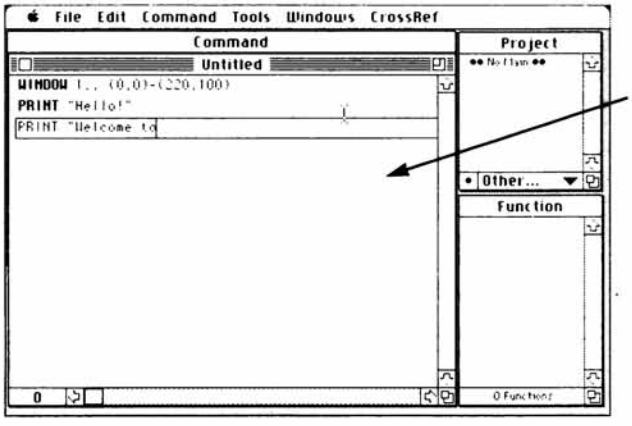
6. Make sure *Check for string/array overflow* is checked.
7. Click the *Remember settings* button so you don't have to repeat these steps every time you run *FutureBASIC*.
8. Click the *OK* button to close the *Preferences* window.

Steps for doing Exercises

Follow the steps below when doing exercises:

1. Read the explanation at the start of each exercise.

2. Type the exercise into an *Untitled* window. If necessary, choose *New* from the File menu to create a new window. Save exercises in their own folder with file names such as; *Ex 1.1, Ex 1.2*, etc. Be sure to save your exercises, as they need to be compared to the answers in the back of the book.



Enter and edit programs in the editor windows (not the Command window)

3. Choose *Run* from the Command menu to test exercises you've entered. If you get an error, fix it, then run the program again until you get it right.

4. When you finish entering all exercises for a lesson, compare your answers to those in the answer section at the back of the book. Your programs don't have to exactly match those in the answer section, as long as your results do.

Steps for doing Programming problems

Follow the steps below when doing Programming problems at the end of each lesson:

1. Before typing in a program, *carefully* read the instructions. Every detail is important. Don't start writing a program until you fully understand what you are being instructed to do.

2. Think about what you've learned in the current and previous lessons and how it can be applied to solving the Programming problem.

3. Type in your program. Test it by choosing *Run* from the Command menu. Does it work as expected? If not, edit any lines that are not working. Run it again, test it again, edit it again, and so on, until the program functions exactly as described.

4. When you've completed the Programming problems, compare your programs to the ones in the answer section of the book. They need not match exactly, but should do everything required.

You can learn a lot from the answer programs in the back of the book. How did the answer program solve the problem? Did your program take more or less lines then the answer program? Did your program use the same methods used by the answer program? What can you do to improve your original program?

Lesson 0
Entering programs

Entering and editing programs is much like typing and editing text in a simple word processor.

When you choose *Run* from the Command menu, *FutureBASIC* translates commands in your program into something the computer understands.

This lesson goes over some of the problems you'll encounter entering programs:

- How to run programs you've entered.
- What to do when you get an error message.
- How to "stop" a program to examine what the program has put in a window.

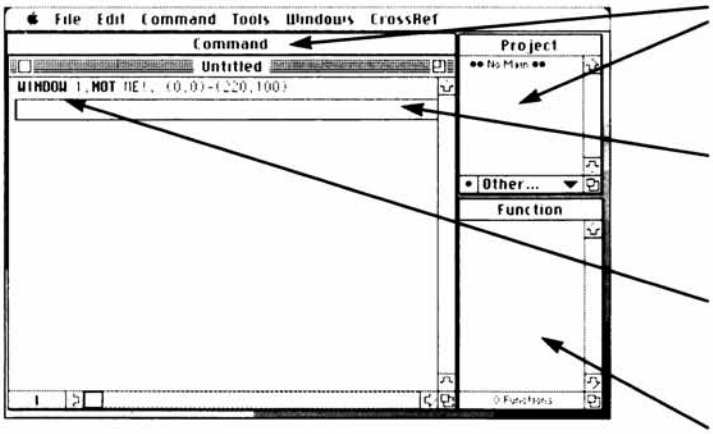
If you haven't done so already, be sure to set FutureBASIC preferences as described on page seven.

Exercise 0.1

Type the program line below into the *Untitled* window. Normally, you'll want to try out programs you enter by running them. For this exercise, wait until the next page.

```
WINDOW 1,NOT ME!,(0,0)-(220,100)
```

This is the program line you will type in. Exercises you will enter are in *Courier* font. When you get to the end of a line, press the return key. Later, when you run the program, it will open a window centered on the screen.



For the purposes of this book, the *Command* and *Project* windows will not be needed. You may ignore them.

Enter exercises and programming problems into the *Untitled* window. If you don't see one, choose *New...* from the File menu.

After you enter a line and press the return key, *FutureBASIC* will highlight all commands it understands.

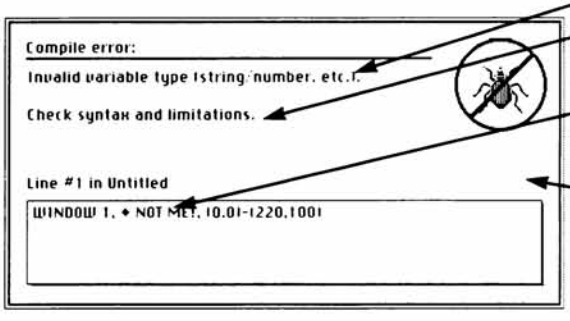
This window will be explained in lesson 12.

Exercise 0.2

To run the program line you entered in *Exercise 0.1*, choose *Run* from the Command menu.

You will get an error message like the one below. *The "NOT ME!" part of the line was added intentionally so you can see how to diagnose and correct errors. All programmers, expert and amateur alike, get errors like this all the time.*

Please read all the notes with arrows for an explanation of what's in the error window and how to return to the editor.


Compile error:
Invalid variable type (string/number, etc.).
Check syntax and limitations.
Line #1 in Untitled
WINDOW 1, ◆ NOT ME!, 10,01-1220,1001

A description of the error.

A possible solution to the error. Check syntax means "Did you type it in correctly?"

This is a copy of the line where the error occurred. The diamond is positioned just before the part of the line that caused the error.

Close the error window by pressing a key or clicking it with the mouse.

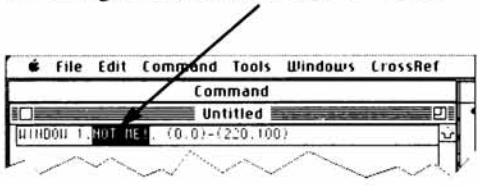
After closing the error window, the line where the error occurred is highlighted.

If you forget what the error was, choose "Show Last Error..." from the Edit menu and the error window will be displayed again.

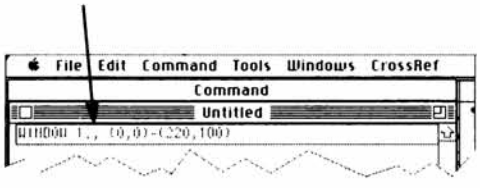
Exercise 0.2 *(continued)*

To fix the program so it will run without an error, edit the program line following the steps below:

1. Using the mouse, select: NOT ME!:



2. After selecting the text, press the delete key to remove it. The line should now look like this:



3. Choose *Run* from the Command menu. The program should execute without giving you an error. If you do get an error, edit the line to look *EXACTLY* like the one above.

   Still getting errors? Here are some things to look for: Did you put a space after WINDOW? Are commas, parentheses and hyphens in the right place? Did you accidentally enter the letter "O" instead of a zero for window coordinates?

4. If, after running the program, you see a bunch of windows opening and closing. *Congratulations!* Your one-line program is working!

When FutureBASIC reaches the end of the program, it closes the program window, and returns to the editor. This happens so fast, that the window created by this program barely has time to appear before it is closed.

The next page explains how to pause a program so you have time to see what is happening.

Exercise 0.3

As you saw in *Exercise 0.2*, the program ran, but quit so fast you didn't have time to examine the window that was opened.

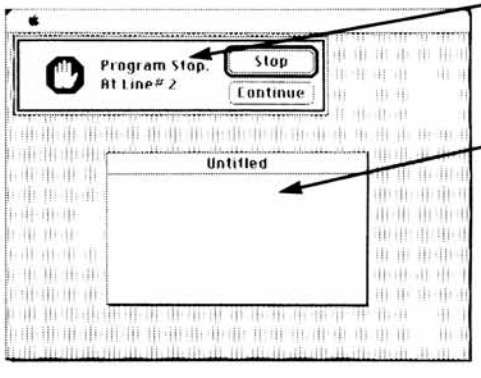
One way to make a program pause before it quits is to add STOP to the last line of your program, as shown below:

```
WINDOW 1,, (0,0)-(220,100)
STOP
```

The (0,0) part of the WINDOW statement is what centers the window in the middle of the screen. See WINDOW statement in the *Reference* manual for more information about windows.

Here's what this program does when you choose *Run*:

1. *FutureBASIC* closes all the editor windows, hides its menu bar, and compiles the program in memory. If no errors are encountered, it runs the program.
2. Line one of the program opens a window centered in the middle of the screen. The window size is 220 screen dots across by 100 screen dots high.
3. The second line halts the program and displays the *Stop* window. This gives you a chance to examine what kind of window was produced by line one.
4. Click the *Stop* button and the program will return to the editor immediately. Click the *Continue* button and the program will continue on the next line following the STOP statement. Since there are no more lines in this program (STOP is the last line), clicking *Continue* produces the same results.



The window opened by the program when it encountered the STOP statement on the second line.

The window opened by the program when it encountered the WINDOW statement on the first line.

Lesson 1
Printing to a window

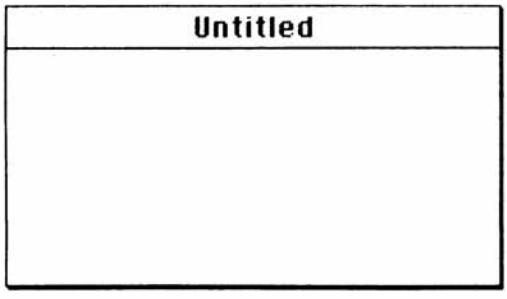
This lesson examines how PRINT is used to send characters to a window. The exercise programs open a small window, print some text and then stop, so you can examine window contents.

How to print text at a specific window position is also covered.

Exercise 1.1

Type in the program, run it, and write what is missing in the window:

```
WINDOW 1,,(0,0)-(210,100)

PRINT "Hello!"
PRINT "Welcome to FutureBASIC"

STOP
```



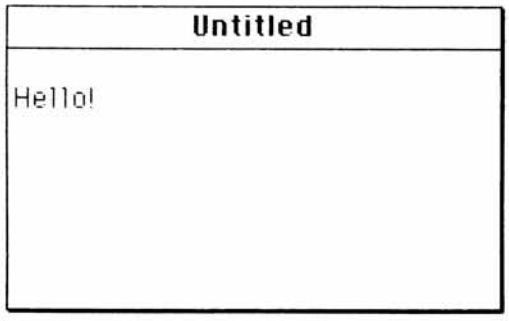
Blank lines make programs easier to read.

When you get an error, edit the offending line to exactly match the exercise text. Check that letters are typed in using the same upper and lowercase. Check that zeros are entered for numbers, not the letter "O" (and visa-versa). Also make sure all the commas, parentheses, and other symbols are entered as shown in the exercise.

Exercise 1.2

Type in the program, run it, and write what is missing in the window:

```
WINDOW 1,,(0,0)-(210,110)

PRINT
PRINT "Hello!"
PRINT "Welcome to FutureBASIC"
PRINT
PRINT "Programming is fun!"

STOP
```

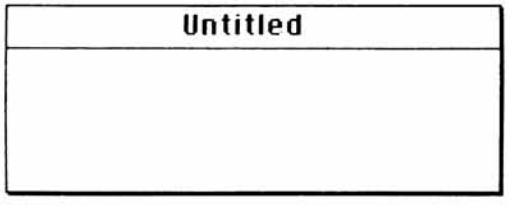


Prints a blank line so the text isn't at the top of the window.

Exercise 1.3

Type in the program, run it, and write what is missing in the window:

```
WINDOW 1,,(0,0)-(210,60)

PRINT
PRINT "one, ";
PRINT "two, ";
PRINT "three"

STOP
```



A semicolon at the end of a print line makes the next `PRINT` statement start printing immediately after it, instead of printing on the next line.

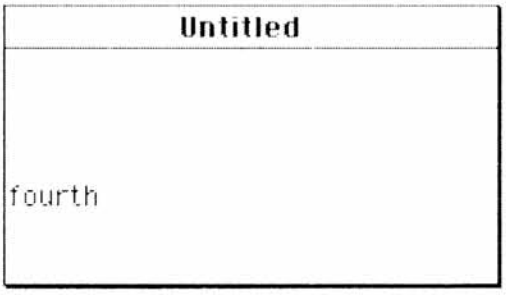
Exercise 1.4

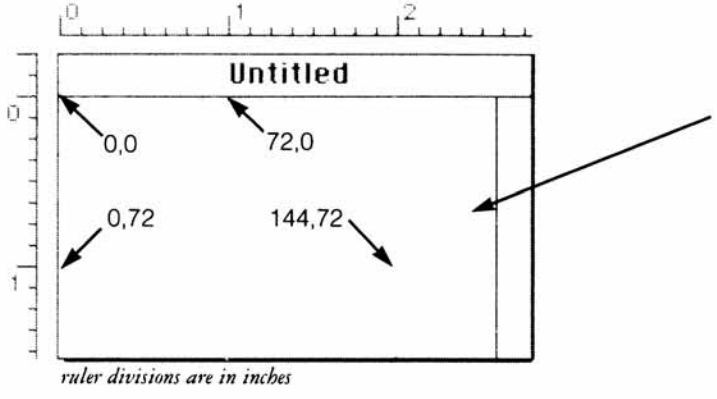
To specify an exact position where text is to be printed, horizontal and vertical coordinates are given after `PRINT` in this format: `PRINT %(h,v)`

The *h* represents the horizontal coordinate. The *v* represents the vertical coordinate.

Type in the program, run it, and write what is missing in the window:

```
WINDOW 1,,(0,0)-(210,100)

PRINT %(0,36) "third"
PRINT %(0,12) "second"
PRINT %(0,24) "first"
PRINT %(0,48) "fourth"

STOP
```





Think of window coordinates as an invisible grid.

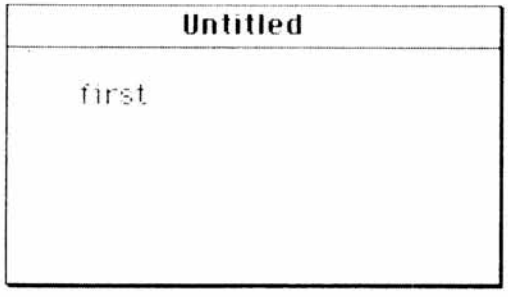
Since there are about 72 pixels per inch on the screen, you should be able to guess approximate pixel positions.

For example, if you want to print at the top of the window, about three inches over from the left, the coordinate would be about (210, 12).

The vertical coordinate positions text for printing at its baseline, not at the top of the text as you might expect.

Exercise 1.5

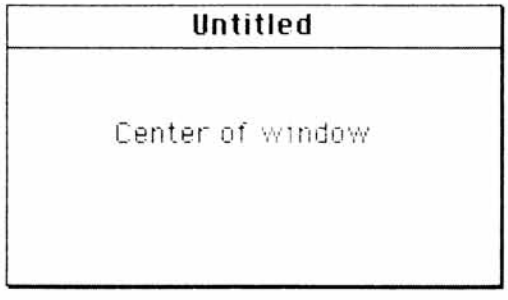
Type in the program, run it, and write what is missing in the window on the right.




```
WINDOW 1,,(0,0)-(210,100)

PRINT %(60,36) "third"
PRINT %( 0,12) "second"
PRINT %(30,24) "first"
PRINT %(90,48) "fourth"

STOP
```

Exercise 1.6

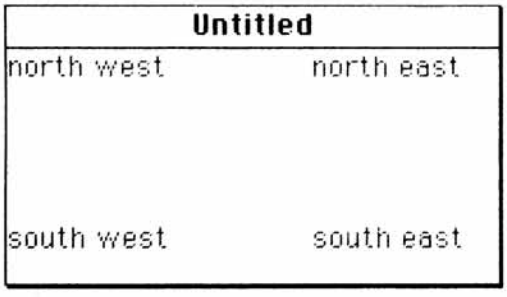
Fill in the missing line below to make the program match the window.




```
WINDOW 1,,(0,0)-(210,100)


STOP
```

Exercise 1.7

Fill in the missing lines below to make the program match the window.

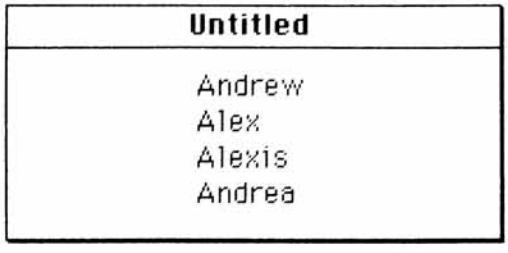



```
_____

_____

_____

PRINT  %(140,85) "south east";

STOP
```

Printing too close to the bottom of a window forces window contents to be scrolled (shifted up one text line). To suppress scrolling, put a semicolon after lines printing close to the bottom of the window, or increase window height.

Exercise 1.8

Fill in the missing lines below to make the program match the window.




```
_____

_____

_____
```

Programming problems

Program problem 1.1

Write a program that will open a window, print your name on the first line, your address on the second line and your city, state and ZIP on the third. The program should stop at the end so you can examine the window contents.

Program problem 1.2

Write a program that will open a window, print Alabama, Alaska, Arizona, and Arkansas lined up in a column down the middle of the window, and then stop so you can examine the window contents.

Lesson 2
Math operations

This lesson examines methods of calculating and displaying numbers. BASIC uses the following math symbols:

symbol	*math operation*
+	addition
-	subtraction
*	multiplication
/	divide

This lesson also introduces the REM statement (short for remark). Any text following a REM statement is ignored, allowing you to enter notes and descriptions about a program line.

Most exercises in this book use an apostrophe ('), a shorthand version of REM, because:

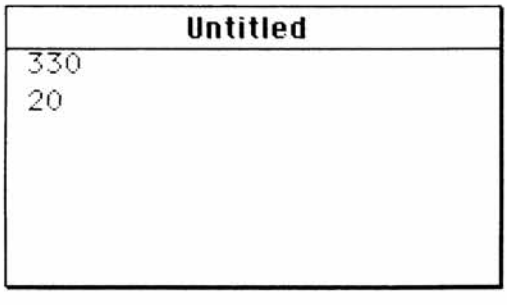
1. One character is easier to type than three
2. They're aligned automatically by *FutureBASIC*
3. They're easier on the eyes than REM statements

Use the straight apostrophe next to the return key (not angled or curly quotes).

Exercise 2.1

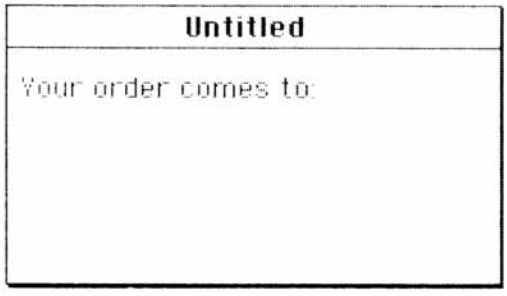
Type in the program, run it, and write what is missing in the window:

```
WINDOW 1,,(0,0)-(210,100)

PRINT 15 * 22   'results of 15 times 22
PRINT 10 + 10   'sum of 10 plus 10
PRINT 14 - 8    '14 minus 8
PRINT 13 / 34   '13 divided by 34

STOP            'halt program
```



Exercise 2.2

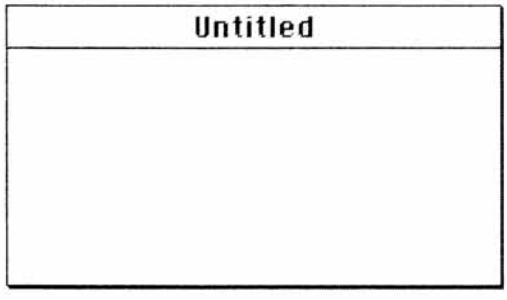
Type in the program, run it, and write what is missing in the window:




```
WINDOW 1,,(0,0)-(210,100)

PRINT
PRINT "Your order comes to:"
PRINT "$7.00 + 7% tax =";
PRINT 7.00 * 1.07       'amt plus tax
PRINT "Thank you for your order"

STOP
```

Exercise 2.3

Type in the program, run it, and write what is missing in the window:

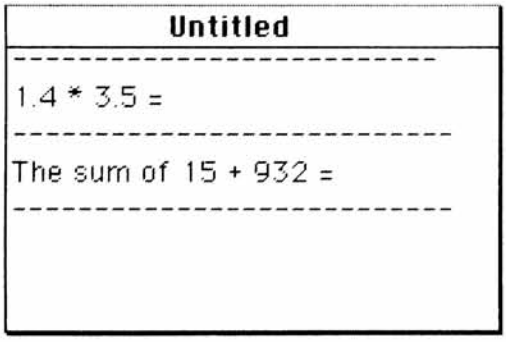



```
WINDOW 1,,(0,0)-(240,120)

PRINT
PRINT "100 divided by 4 + 10 =";
PRINT 100 / 4 + 10     'mix divide/add
PRINT
PRINT "3000 times 3045 =";
PRINT 3000 * 3045

STOP
```

Exercise 2.4

Type in the program, run it, and write what is missing in the window:

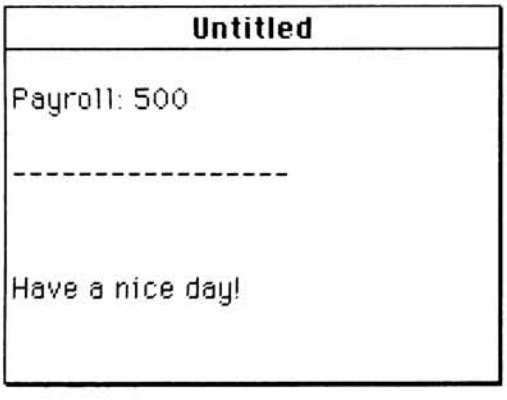
```
WINDOW 1,,(0,0)-(210,120)

PRINT "-------------------------"
PRINT "1.4 * 3.5 = ";
PRINT  1.4 * 3.5

PRINT "-------------------------"
PRINT "The sum of 15 + 932 = "15 + 932
PRINT "-------------------------"

STOP
```



Exercise 2.5

Type in the program, run it, and write what is missing in the window:

```
WINDOW 1,,(0,0)-(210,140)

PRINT
PRINT "Payroll: 500"
PRINT "minus taxes:"500*.3
PRINT "-----------------"
PRINT "Net pay:" 500-500*.3

PRINT
PRINT "Have a nice day!"

STOP
```



Exercise 2.6

Look at the window on the right and fill in the missing program lines below:

```
WINDOW 1,,(0,0)-(210,200)
PRINT
_____________________________________
_____________________________________
_____________________________________
PRINT "2+5*70 equals" 2+5*70
PRINT
_____________________________________
_____________________________________
_____________________________________
_____________________________________
PRINT "(2+5)*70 equals" (2+5)*70

STOP
```

Untitled

Multiply and divide are done
BEFORE add and subtract,
sometimes giving you un-
expected results:
2+5*70 = 352

Use parentheses to force the
order of operations, so
answers are calculated as
you expect:
(2+5)*70 = 490

Exercise 2.7

Look at the window on the right and fill in the missing program lines below to match it:

```
_____________________________________
_____________________________________
_____________________________________
_____________________________________
_____________________________________
_____________________________________
```

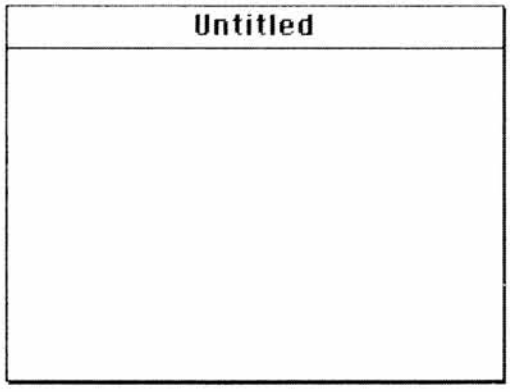
Untitled

Answer with no parentheses:
15 + 20 * 5 = 115

Answer with parentheses:
(15 + 20) * 5 = 175

Exercise 2.8

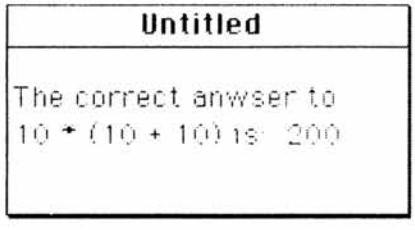
Type in the program, run it, and fill in what is missing in the window:




```
WINDOW 1,,(0,0)-(210,150)

PRINT
PRINT "Payroll calculations"
PRINT "--------------------"
PRINT "Mike, 10 h @ $9 hr:" 10 * 9
PRINT "Gary, 10 h @ $4 hr:" 10 * 4
PRINT "--------------------"
PRINT "Total payroll      :" 10*9+10*4

STOP
```

Exercise 2.9

Look at the window on the right and fill in the missing program lines below to match it:




```
WINDOW 1,,(0,0)-(170,70)


PRINT "The correct answer to"
```

Programming problems

Program problem 2.1

Write a program that prints the names, hours and total pay (at $9.00 per hour) of the following people.

John: 35.5 hrs, Mary: 26.9 hrs, Larry: 22.3 hrs

Program problem 2.2

Write a program that prints a title called "Order of Operations" and a line with the problem "500 + 200 * 23 =", along with the result.

Make sure the program calculates 500 + 200 first and then multiplies the result by 23.

Use parentheses if necessary to force the order of operations to return the expected answer.

FutureBASIC supports a number of other math functions. If you're interested in math and scientific functions, see the Math appendix in the Reference manual.

Lesson 3
Text characteristics

This lesson uses the TEXT statement to change text characteristics for the current window:

```
TEXT font, point size, style
```

Any text printed after the TEXT statement will use the font, size and style specified.

Some constants and values used to specify font, point size and character style include:

font constants	*point sizes*	*style constants*	
_newYork	9	_boldBit%	(bold faced)
_geneva	12	_uLineBit%	(underlined)
_monaco	18	_italicBit%	(italicized)

These are just a few of the constants available for setting text characteristics. See the TEXT statement in the *Reference* manual for a list of others.

To print the word "Hello" in Geneva font, 12 point size, and underline style, use:

```
TEXT _geneva, 12, _uLineBit%
PRINT "Hello"
```

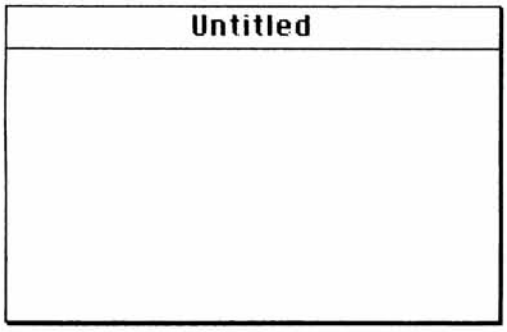
These words are called *constants* because they represent a constant, unchanging value. Constants make a program easier to read. These constants are part of a predefined list of constants *FutureBASIC* understands. You may also define your own constants. See the *Constants appendix* in the *Reference* manual for more information.

Constants must be preceded by an underline character. Be sure to use the % sign after style constants.

Exercise 3.1

Type in the program, run it, and write what is missing in the window:

```
_plainText = 0        'define plain constant

WINDOW 1,,(0,0)-(210,110)

TEXT _geneva, 9, _plainText
PRINT
PRINT "My name is:  ";

TEXT _newYork, 12, _boldBit%+_ulineBit%
PRINT "Mr. Michael Smith!"

STOP
```

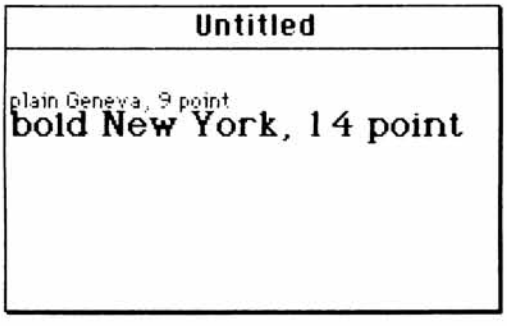


This defines a constant called "_plainText" which is used to make unstyled text (a value of zero). A constant can only be defined once in a program. It is customary to define constants together at the top of a program so they can be located easily.

Exercise 3.2

Look at the window and fill in the missing line below:

```
_plainText = 0        'define plain constant

WINDOW 1,,(0,0)-(210,110)

PRINT
TEXT  _geneva, 9, _plainText
PRINT "plain Geneva, 9 point"

_____________________________________
PRINT "bold New York, 14 point"

STOP
```



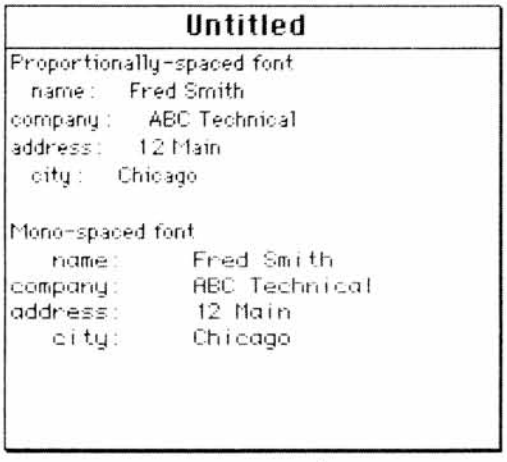
Exercise 3.3

It is important to know the difference between *mono-spaced* and *proportionally-spaced* fonts.

Each character in a mono-spaced font has the same width. Monaco and Courier are two commonly used mono-spaced fonts. It's easier to align text with mono-spaced fonts.

Each character in proportionally-spaced fonts has its own width. For example, a "W" is twice as wide as a "v". Most fonts are proportionally-spaced because they are more readable and more characters fit on a line.

Look at the window and fill in the missing program lines below:

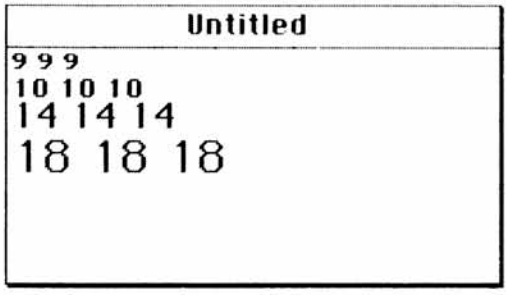



```
_plainText = 0        'Define plain constant

WINDOW 1,,(0,0)-(210,170)

TEXT  _geneva, 9, _plainText
PRINT "Proportionally-spaced font"
PRINT "   name:      Fred Smith"
PRINT "company:      ABC Technical"
PRINT "address:      12 Main"
PRINT "   city:      Chicago"
PRINT

TEXT  _monaco
PRINT "Mono-spaced font"
PRINT "   name:      Fred Smith"



STOP
```

If a parameter is not specified, the previously set parameter is used. This changes the font to Monoco, but leaves the size at 9 and the style as plain, as specified in the first TEXT statement.

Exercise 3.4

Look at the window and fill in the missing program lines below:




```
WINDOW 1,,(0,0)-(210,100)

TEXT  _geneva, 9, _boldBit%
PRINT "9 9 9"

TEXT ,10
PRINT "10 10 10"

TEXT ,14

_____ _____

PRINT "18 18 18"

STOP
```

Exercise 3.5

Look at the window and fill in the missing program lines below:

```
_normalText = 0    'define normal text

WINDOW 1,,(0,0)-(200,260)

TEXT _newYork,18, _boldBit%
PRINT %(2,20) "A True Story..."
```

_____

_____

_____

_____

_____

_____

_____

_____

_____

```
STOP
```

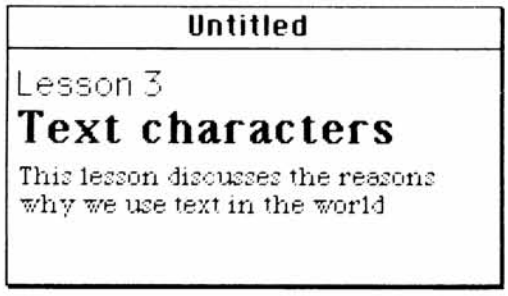
Untitled
A True Story...
A long time ago, there was a little girl named Goldilocks who had long, blond hair.
One day she decided to go for a walk in the forest. Before long, she came upon a little cottage with pretty flowers all around it.
"I think I'll go in for a look." she said to herself.
page 1

To put double-quotes and apostrophes into text, so they're not interpreted as program quotes or remarks, use the curly quotes. The keystrokes for getting these characters are:

character	*keystrokes*
“	option - [
”	shift-option - [
‘	option -]
’	shift-option -]

Exercise 3.6

Look at the window and fill in the missing program lines below:




```
_normalText = 0    'define normal text

WINDOW 1,,(0,0)-(210,100)

TEXT  _geneva, 14, _normalText
```

_____

```
TEXT _newYork,18, _boldBit%
```

_____

_____

_____

```
STOP
```

Programming problems

Program problem 3.1

Write a program that prints a column of years: 1992, 1993, 1994, 1995 in Geneva, 9 point, plain text. At the top of the column put the word "Years", in Geneva, 12 point, underlined, bold text.

Program problem 3.2

Write a program that prints the word "Hello" in 9, 18, 24, and 36 point, Helvetica font. Put a space between each printed line.

Program problem 3.3

Write a program that prints a column containing these words in proportional, bold text: *Monday, Tuesday* and *Wednesday*.

About a half-inch to the right of each day, print another column of proportionally-spaced, plain text that says *Holiday, Work day* and *Work day*. Words in each column must line up.

Lesson 4

An introduction to graphics

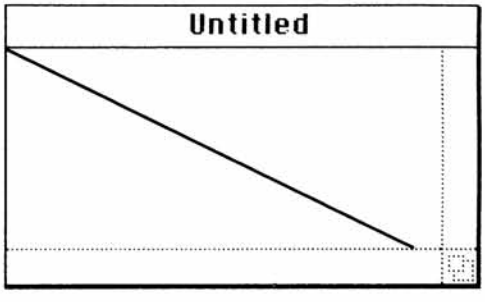
This lesson uses the PLOT, PEN, and BOX statements to draw in a window.

When drawing graphics, use the same coordinate system you used in lesson three with PRINT %(h,v).

To draw a line from the upper-left corner of the window down to the opposite corner, specify points that coincide with window dimensions, like the ones in the PLOT statement below:

```
WINDOW 1,, (0,0)-(200,100)
PLOT 0,0 TO 200,100
STOP
```

The result of these three lines would look like this:



The reason the line doesn't go all the way to the corner is because *FutureBASIC* allocates space for a grow box and scoll bars in default window types. It assumes you will be using the scroll bars and don't want them over text or graphics.

See *Exercise 4.1* for a way of creating windows without a grow box or scroll bars.

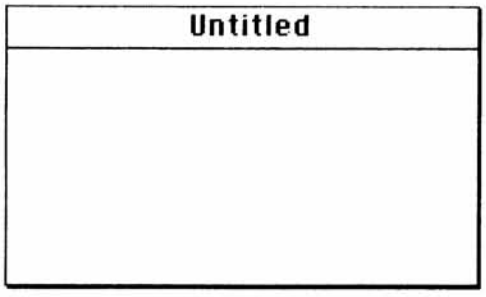
Exercise 4.1

There are many window types. Take a look at the WINDOW statement in the *Reference* manual for a list of those available.

To specify a window type without a grow box or scroll bars, put _docNoGrow at the end of the WINDOW statement (see below).

Type in the program, run it, and write what is missing in the window:

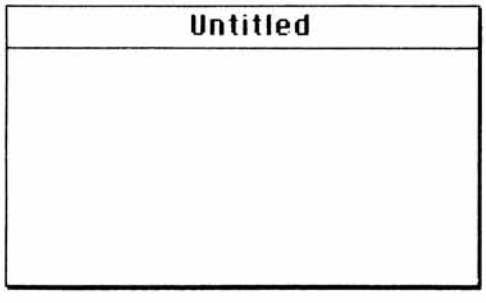
```
WINDOW 1,,(0,0)-(200,100), _docNoGrow

PLOT 0,0 TO 200,100
BOX 10,10 TO 190,90

STOP
```



Exercise 4.2

Type in the program, run it, and write what is missing in the window:

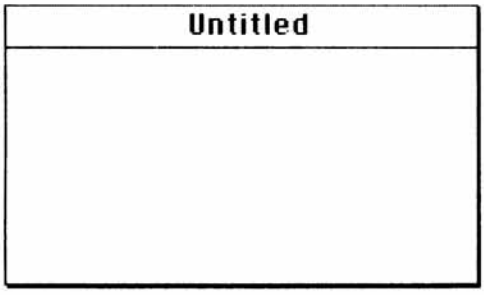
```
WINDOW 1,,(0,0)-(200,100), _docNoGrow

PLOT 10,50 TO 170,0
PLOT TO 170,100 TO 10,50

STOP
```



The TO statement used with PLOT or BOX, tells *FutureBASIC* to draw from the *last* position *to* the *new* position. More than one TO statement can be used to add additional points to a line.

Exercise 4.3

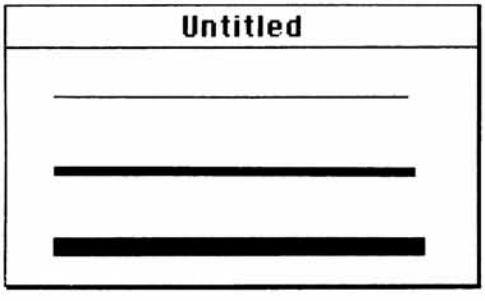
The PEN statement is used to change the thickness of lines, set line or fill patterns, and other things. Type in the program, run it, and write what is missing in the window:




```
WINDOW 1,,(0,0)-(200,100), _docNoGrow

PLOT 0,0 TO 10,80

PEN 4,4
PLOT 5,0 TO 100,80

PEN 8,8
PLOT 20,0 TO 200, 80

STOP
```

Exercise 4.4

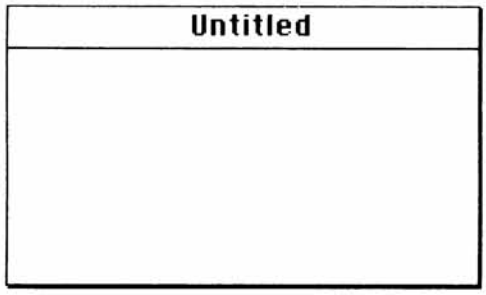
Look at the window and write what is missing in the program lines below:




```
WINDOW 1,,(0,0)-(200,100), _docNoGrow

PLOT 20,20 TO 170,20
PEN 4,4

_____________________________________
PLOT 20,80 TO 170, 80

STOP
```

Exercise 4.5

Text and graphics can be mixed easily. Type in the program, run it, and write what is missing in the window:

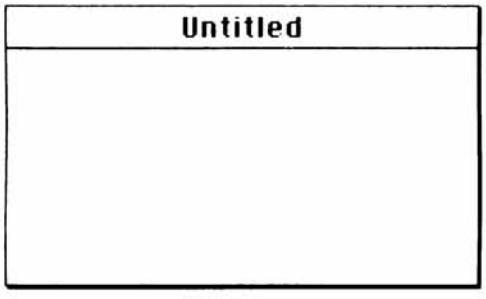



```
WINDOW 1,,(0,0)-(200,100), _docNoGrow

BOX 20,20 TO 180,80
PRINT %(43,53) "Text and graphics"
PEN 3,3
BOX 30,30 TO 170,70

STOP
```

Exercise 4.6

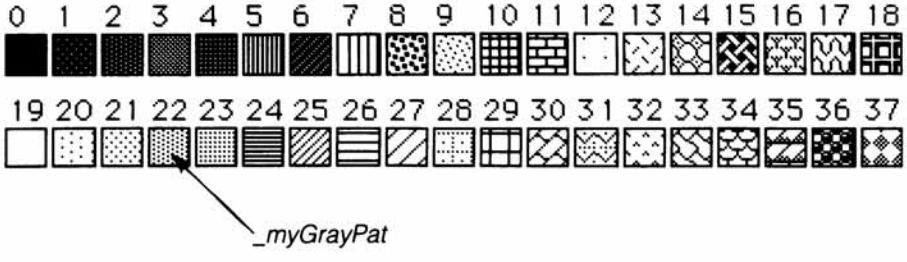
You can change the pen pattern with the fifth parameter in the PEN statement. BOX FILL draws a box filled with the current pen pattern.

Type in the program, run it, and write what is missing in the window:



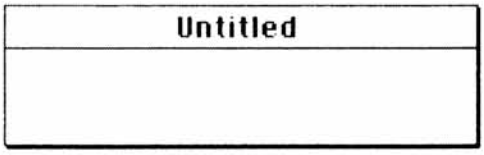

```
_myGrayPat = 22              'see chart below

WINDOW 1,,(0,0)-(200,100), _docNoGrow

PEN 2,2                      '2 pixel pen
BOX 10,10 TO 190, 90

PEN ,,,,_myGrayPat           'gray pattern
BOX FILL 12,12 TO 188,88     'filled box

STOP
```

If no parameter is specified (a comma with nothing before it) the previously defined parameter is used. For example, this line uses a pen width and height of 2,2, as defined on the third line.



Exercise 4.7

Type in the program, run it, and write what is missing in the window:

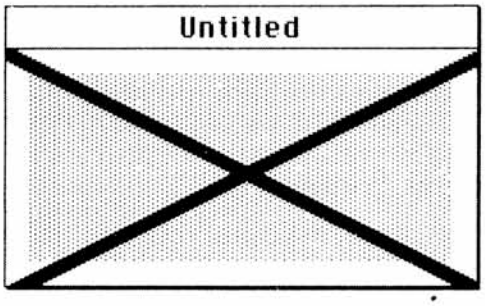



```
'----- Constants -------
_solid  = 0
_gray1  = 1
_gray2  = 2
_gray3  = 3

WINDOW 1,,(0,0)-(200,40), _docNoGrow

PEN 5,5,,,_solid            'black
PLOT 10,10 TO 185,10

PEN ,,,,_gray1              'dark gray
PLOT 10,15 TO 185,15

PEN ,,,,_gray2              'lighter gray
PLOT 10,20 TO 185,20

PEN ,,,,_gray3              'even lighter
PLOT 10,25 TO 185,25

STOP
```

A common error is entering too few or too many commas, or putting commas in the wrong place. In this example, patterns should be from dark to light. If not, check commas.

Exercise 4.8

Look at the window and fill in the missing lines in the program below:



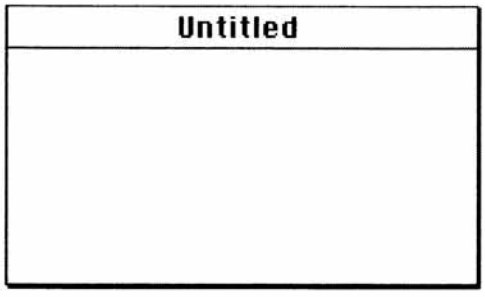

```
_myGray  = 22

WINDOW 1,,(0,0)-(200,100), _docNoGrow

PEN 1,1,,,_myGray          'light gray
```

Exercise 4.9

The TEXT statement allows you to control how text is drawn. Among other things, this permits drawing white text on a black background.

Type in the program, and fill in what is missing in the window:

```
WINDOW 1,,(0,0)-(200,100), _docNoGrow

BOX· FILL 10,10 TO 190,90

TEXT _geneva, 12, _boldBit%, _srcXOR
PRINT %(42,50) "White on black!"

STOP
```



Copy modes are explained more in the *Pen* section of *Basic drawing tools* chapter of the *Handbook*. Also see TEXT in the *Reference* manual for more information.

See TEXT in the Reference manual for more examples of printing text on backgrounds. If you're interested in graphics, take a look at the QuickDraw section of the Handbook.

Programming problems

Program problem 4.1

Write a program that prints the word "Graphics" (Geneva, 12 point, bold) in the center of the window. Draw two lines–each two pixels thick–under the word. Keep the lines 5 pixels apart.

Program problem 4.2

Write a program that prints the word "HELLO" in the middle of a window (Geneva, 18 point, plain). Then draw two lines connecting opposite corners of the window, making a big "X" across the window and over the word. The output window must have round corners. Hint: See the WINDOW statement in the *Reference* manual.

Lesson 5

Variables for holding numbers

So far, all the exercises have used numbers, quoted strings or unchanging constants. *FutureBASIC* allows you to store values in a word or letter, called a *variable* (so-called because the value can vary).

If you're familiar with algebra, you know that if A=10 and B=20, then A+B equals 30. Variables work in a similar way.

If you have a variable called *velocity*, you can assign it the value 2,345 by using the statement:

```
velocity = 2345
```

A statement like this, that has a variable followed by an equal sign and a number, is called an assignment statement.

A statement like this will cause the variable to be assigned the last value assigned to it.

Some types of variables for holding numbers:

Integer — Integer variables hold whole numbers like 3200, 3 and -4565. Range is plus or minus 32767.

Integer variable is assumed if you don't use a symbol, but a percent sign may be used. Examples: *MPH*, *years*, age, *months*, *daysOfTheMonth*, *secondsInAnHour%*.

Long integer — Long integer variables hold whole numbers like 39430, -345777, 2000000 and 5. Range is plus or minus 2 billion.

Long integer variables must end with an ampersand (&). Examples: *speedOfLight&*, *distanceToSun&*, *weightOfTheEarth&*, *secondsInDay&*.

Floating point — Floating point variables hold real numbers with fractions like: 26.341, 3.5 and .879.

Range is very large with about 8 digits of precision. Digits of precision can be set in *Preferences* under the Edit menu.

Floating point variables must end with an exclamation mark (!). Examples: *taxRate!*, *weightOfAFly!*, *mphOfASnail!*, *grossNationalProduct!* and *costInDollars!*

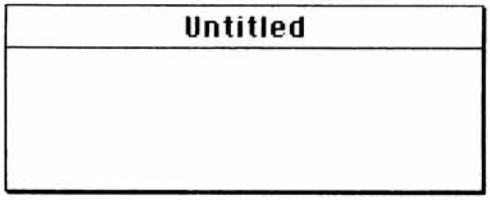
Case is significant in variable names. For example, the variable "wide" is not considered the same variable as "Wide". This is a common error to look for when variables return unexpected values.

If you assign an integer or long integer variable a floating point value, such as 2.5, the fractional portion will be lost and the integer variable will equal the whole value (2 in this case). This is common mistake you should look for if values that should have a fraction are being returned as a whole number.

Exercise 5.1

Type in the program below, run it, and write the result in the window.

```
WINDOW 1,,(0,0)-(200,100)

wide   = 5
height = 6
area   = wide * height

PRINT %(60,30) "Area=";area

STOP
```



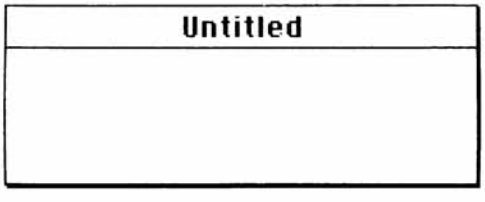
The space you see when this variable prints is because numbers and numeric variables always print with a leading space (reserved for a minus sign when printing a negative number).

Exercise 5.2

Use as many letters as necessary to make descriptive variable names. Descriptive variables are much easier to understand than one or two letter variables. Imagine trying to understand the program below if the letters *p!* and *h* were used instead of *payRate!* and *hours!*

Type in the program below, run it, and write the results in the window.

```
WINDOW 1,,(0,0)-(200,100)

payRate! = 5.5          'hourly rate
hours!   = 23.5         'hours worked

PRINT                   'blank line
PRINT "Pay =" payRate! * hours!

STOP
```



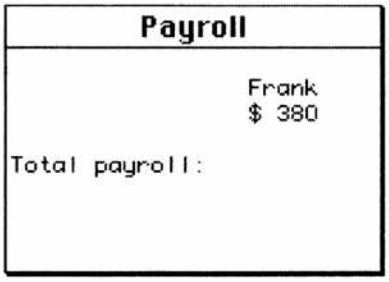
Since variable names cannot have spaces, many people capitalize the second and subsequent words to make them easier to read. Some people capitalize the first letter as well.

Examples: myWorkTaxes!, corpLawRates!, farmSubsidies!, DaysOfSickLeave%, etc.

Exercise 5.3

Type in the program below, run it, and write what is missing in the window:

```
WINDOW 1,"Payroll",(0,0)-(160,95)

hours  = 40
pay!   = 9.50
frank! = hours * pay!

hours  = 39
pay!   = 12.75
mary!  = hours * pay!

TEXT _monaco, 9
PRINT
PRINT " Mary", " Frank"
PRINT " $"mary!, " $" frank!
PRINT
PRINT "Total payroll: $" mary!+frank!

STOP
```



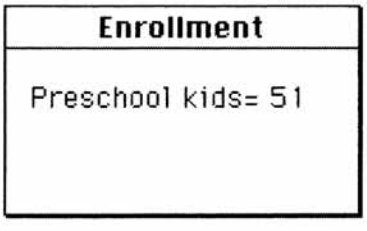
This adds a title to the window.

Commas are used to *tab* over to the next tab position. The default tab positions are every 16 characters. See the DEF TAB statement in the *Reference* manual for more information.

Exercise 5.4

Look at the window and fill in what is missing in the program below:

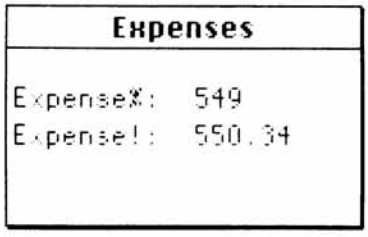
```
WINDOW 1,"Enrollment",(0,0)-(150,70)

toddlers  = 29
babies    = 22

_____________________________________
PRINT
PRINT "  Preschool kids=" enrollment

STOP
```



A very common error when entering programs is misspelling a variable name or using different case in a letter. Check variables for this if errors occur.

Exercise 5.5

Assigning floating point values to integer variables will result in the fraction being lost. Look at the window and fill in what is missing in the program below:

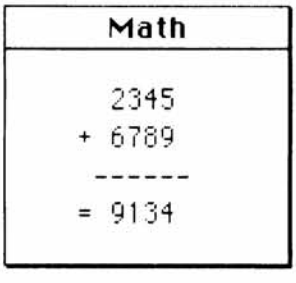



```
WINDOW 1,"Expenses",(0,0)-(150,75)

rent%      = 523.33
rent!      = 523.33
    .
electric% = 22.45
electric! = 22.45

water%     = 4.56
water!     = 4.56

TEXT _monaco, 12
PRINT

PRINT "Expense!: " rent! + electric! + water!

STOP
```

Line up equal signs and add line spaces to make it easier to read programs and find bugs. Also use blank lines liberally to group related statements.

Exercise 5.6

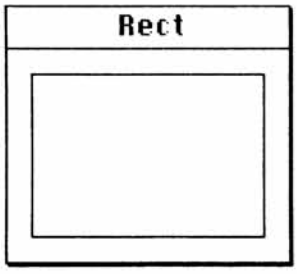
Look at the window and fill in what is missing in the program below:




```
WINDOW 1,"Math",(0,0)-(120,90)

first  = 2345

sum    = first + second

PRINT
TEXT _monaco, 12

PRINT "     +" second
PRINT "      ------"
PRINT "     =" sum;

STOP
```

Exercise 5.7

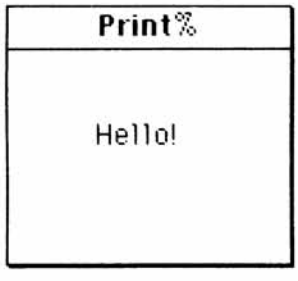
Look at the window and fill in what is missing in the program below:




```
WINDOW 1,"Rect",(0,0)-(120,90),_docNoGrow

top%    = 10
left%   = 10
bottom% = 80
right%  = 110

_____________________________________________

STOP
```

Exercise 5.8

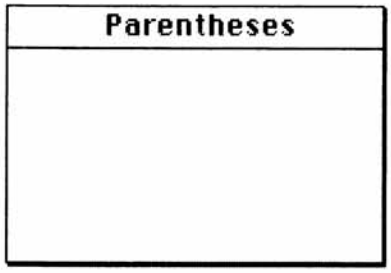
Look at the window and fill in what is missing in the program below:




```
WINDOW 1,"Print%",(0,0)-(120,90),_docNoGrow

_____________________________________________

PRINT %(horz,vert) "Hello!"

STOP
```

Exercise 5.9

Type in the program below, run it, and fill in what is missing in the window:




```
WINDOW 1,"Parentheses",(0,0)-(160,90)

velocity! = 3.456

PRINT
PRINT velocity! * 6 + 2
PRINT velocity! * (6 + 2)

STOP
```

Programming problems

Program problem 5.1

Write a program that calculates and prints ABC Corp's payroll. The company's three employees and their pay:

```
Tom     $123.34
Larry   $560.32
Mary    $455.55
```

Print the employee's name, pay, and total payroll. Use variables in the program. Use mono-spaced characters so names and pay line up. Make the window title "Payroll".

Program problem 5.2

Write a program that makes one integer variable equal to 20,000 and one equal to 22,000 (don't use commas).

Make an integer variable called *sum%* equal to the sum of the two integer variables. Print it in the window (if the result seems wrong, read the note at right).

Also, make a floating point variable called *sum!* equal to the sum of the two integers. Print it to the window also.

Integer variables have a range of -32768 to +32767. Larger or smaller numbers will overflow and be incorrect. Long integer variables hold values in the range of -2 billion to +2 billion. Use long integer variables to hold large whole numbers. Use floating point variables to hold large real numbers. Note: Integer math is much faster than floating point math.

Lesson 6

Variable for holding text characters

A variable type that holds a *string* of text characters, such as "Hello there!", "Welcome", "Have a nice day." and so on, is called a string variable.

String variables end with a "$" and can hold up to 255 characters such as "A" through "Z", "a" through "z", "0" through "9", and symbols like "!@#$%^&*", among others.

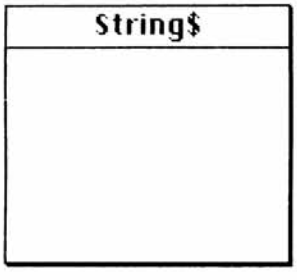
Examples of string variable names: *firstName$, greeting$, answer$, replyToQuestion$, anniversaryDate$.*

This lesson will show you how to store characters in string variables, add string variables together, get characters from the left, right, or middle of string variables, and print string variables.

Exercise 6.1

Type in the program below, run it, and fill in what is missing in the window:

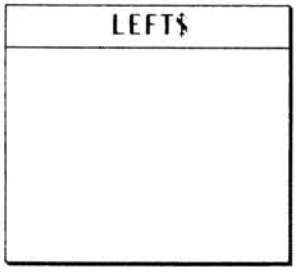
```
WINDOW 1,"String$",(0,0)-(120,90)

firstName$ = "Mike"
LastName$  = "Smith"

wholeName$ = firstName$ + " " + LastName$
PRINT
PRINT wholeName$

STOP
```



This concatenates the three strings (the quoted space is also a string).

Exercise 6.2

Type in the program below, run it, and fill in what is missing in the window:

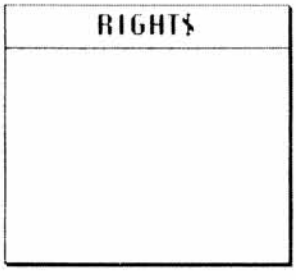



```
WINDOW 1,"LEFT$",(0,0)-(120,90)

sentence$  = "Hello, my name is Mike."
firstWord$ = LEFT$(sentence$,5)

PRINT
PRINT firstWord$

STOP
```

This gets five characters from the left side of a string (string variables or quoted strings).

Exercise 6.3

Type in the program below, run it, and fill in what is missing in the window:

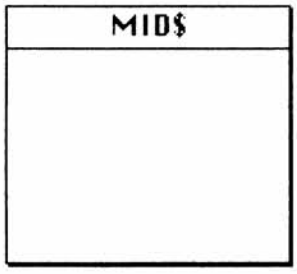



```
WINDOW 1,"RIGHT$",(0,0)-(120,90)

sentence$  = "Hello, my name is Mike."
lastWord$ = RIGHT$(sentence$,5)

PRINT
PRINT lastWord$

STOP
```

This gets five characters from the right side of a string (string variables or quoted strings).

Exercise 6.4

Type in the program below, run it, and fill in what is missing in the window:

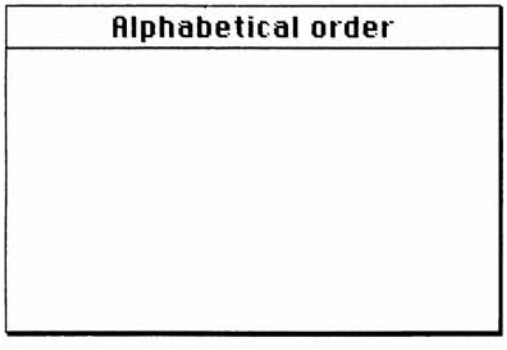



```
WINDOW 1,"MID$",(0,0)-(120,90)

sentence$  = "Hello, my name is Mike."
midWord$   = MID$(sentence$, 11, 4)

PRINT
PRINT midWord$

STOP
```

This gets four characters from the center of a string (string variables or quoted strings). The first character is from the 11th position.

Exercise 6.5

Type in the program below, run it, and fill in what is missing in the window:

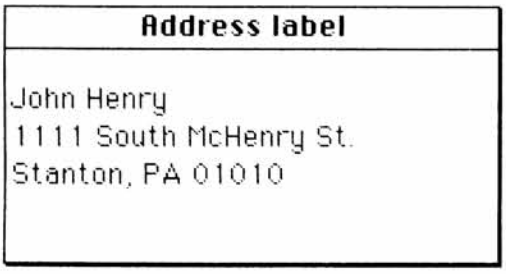



```
WINDOW 1,"Alphabetical order",(0,0)-(210,120)

word1$ = "Pluto"
word2$ = "Moon"
word3$ = "Asteroids"

PRINT
PRINT "Word list:"
PRINT word1$", "word2$", "word3$

PRINT
PRINT "Alphabetical order is:"
PRINT word3$", "word2$", "word1$

STOP
```

Exercise 6.6

Look at the window and fill in the missing program lines below:

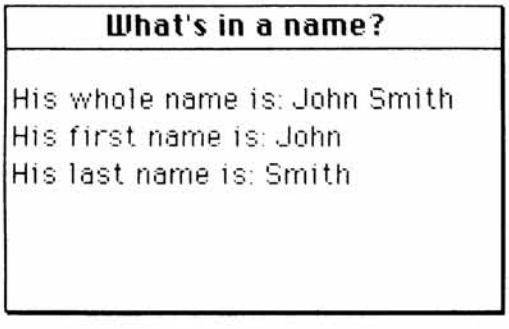
```
WINDOW 1,"Address label",(0,0)-(210,90)

firstName$ = "John"

_____________________________________________
address$   = "1111 South McHenry St."
city$      = "Stanton"

_____________________________________________
zip$       = "01010"

PRINT
PRINT firstName$" "lastName$

_____________________________________________
PRINT city$ ", "state$" "zip$

STOP
```



Exercise 6.7

Type in the program below, run it, and fill in what is missing in the window:

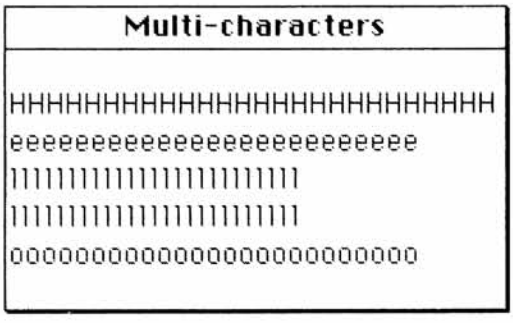
```
WINDOW 1,"What's in a name?",(0,0)-(210,110)

wholeName$ = "John Smith"
firstName$ = LEFT$(wholeName$, 4)

_____________________________________________
PRINT
PRINT "His whole name is: "wholeName$

_____________________________________________
PRINT "His last name is: "lastName$

STOP
```



Exercise 6.8

Look at the window and fill in the missing lines in the program below:

```
WINDOW 1,"multi-characters",(0,0)-(210,110)

TEXT _geneva, 12
PRINT
PRINT STRING$(25,"H");
PRINT STRING$(25,"e")
```

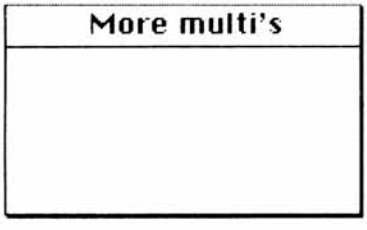


The reason there are more or less characters on some lines is that proportionally-spaced characters have different widths.

Exercise 6.9

Type in the program below, run it, and fill in what is missing in the window:

```
WINDOW 1,"More multi's",(0,0)-(150,70)

TEXT _geneva, 9
PRINT
PRINT "-------------------------"
PRINT STRING$(25,"-")

myHyphens$ = STRING$(25,"-")
PRINT myHyphens$;

STOP
```



Just as you can say the same thing in different ways, you can accomplish the same thing using different programs.

Program problem 6.1

Write a program that puts a book's name in one variable and the author's name in another variable. Then open a window called "Book Card" and put the book's name on the second line and the author"s name on the third line.

Program problem 6.2

Write a program with "Frank Smith" in the variable name1$ and "Mary Jones" in the variable name2$.

Print a heading that says "We are proud to announce the wedding of Frank Smith and Mary Jones". Using `LEFT$`, pull the first names out of the variables and put them into their own variables. Print the two first name variables you created in a sentence that says: "Frank and Mary will exchange vows."

The window title should say "Wedding Announcement".

Lesson 7

Reading data from the keyboard

Programs often need to get information from a user. The INPUT statement gets data entered at the keyboard and stores it in a variable.

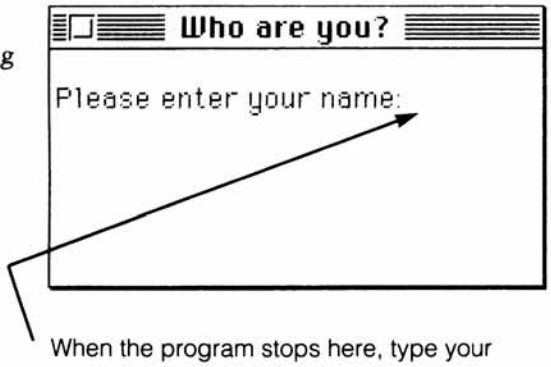
Data entered from the keyboard is stored in variables so your program can manipulate them. Use string variables to retrieve text input and numeric variables to retrieve numeric input.

INPUT will not accept commas or carriage return characters. It also will not allow anything else to be done until data has been typed in and the *return* key is pressed.

Exercise 7.1

Type in the program below, run it, and fill in what is missing in the window:

```
WINDOW 1,"Who are you?, (0,0)-(210,100)

PRINT
INPUT "Please enter your name: ";you$
PRINT "Hello there "you$"!"

STOP
```

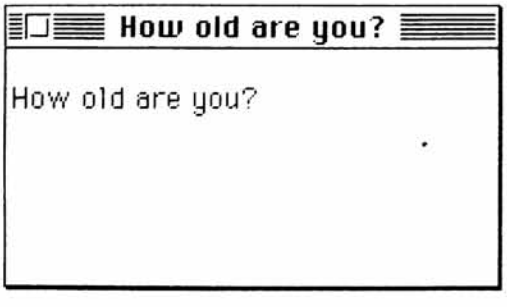


When the program stops here, type your name, and press <return>.

Exercise 7.2

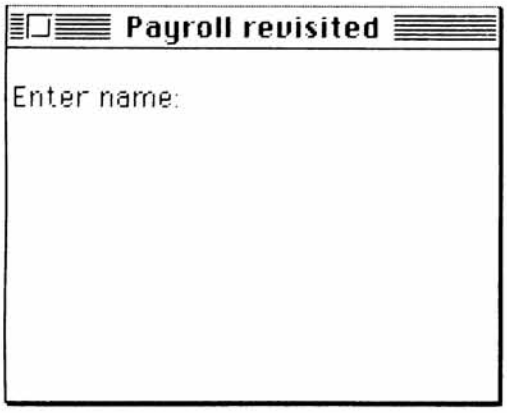
Type in the program below, run it, and fill in what is missing in the window:

```
WINDOW 1,"How old are you?", (0,0)-(220,100)

PRINT
INPUT "How old are you? ";age
PRINT "Wow! You're" age * 12 "months old!"

STOP
```



Exercise 7.3

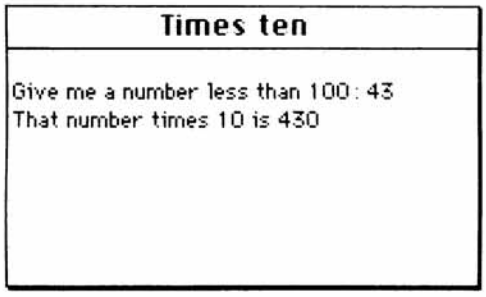
Type in the program below, run it, and fill in what is missing in the window:




```
WINDOW 1,"Payroll revisited", (0,0)-(210,150)

PRINT
INPUT "Enter name: "          ;empName$
INPUT "Enter hours worked: ";hours
INPUT "Enter hourly wage: " ;wage!
PRINT "Employee: "empName$
PRINT "earned $" hours * wage! "this week!"

STOP
```

Exercise 7.4

Look at the window and fill in the missing line in the program below:




```
WINDOW 1,"Times ten",(0,0)-(200,100)

TEXT _geneva, 9
PRINT

_____________________________________
PRINT "That number times 10 is" num% * 10

STOP
```

Programming problems

Program problem 7.1

Get a person's height in feet and inches. Then print how tall the person is in inches.

Program problem 7.2

Get a person's Zip, Address, State, City and Name (in that order), then print the address in mailing label format.

Lesson 8

Comparing values with IF

A program makes decisions by comparing one value to another. One way it can do this is by using IF and LONG IF statements with one of the following symbols:

symbol	*meaning*
=	equal
><	not equal
<>	not equal
>	greater than
>=	greater than or equal to
=>	equal to or greater than
<	less than
<=	less than or equal to
=<	equal to or less than

For example, in the following example, the variable *wage!* is compared to the number 5.5. If *wage!* is equal to 5.5, the line under LONG IF will be printed. If false, it will not be printed.

```
LONG IF wage! = 5.5
  PRINT "Employee makes $5.50 per hour!"
END IF
```

In the example below, if *answer$ = "yes" is true*, the line under LONG IF is printed. If *answer$ = "yes" is false,* the line under XELSE is printed.

```
LONG IF answer$ = "yes"
  PRINT "Thank you for saying yes!"
XELSE
  PRINT "I'm sorry to hear that."
END IF
```

FutureBASIC automatically indents lines between parts of a LONG IF statement to make them easier to read.

You can also use the single-line version of the IF statement for short compares:

```
IF wage! <= 5 THEN tax!=.1 ELSE tax! = .2
```

Use the short form of IF only if the line is short and easy to read. As soon as an IF statement gets too long or hard to read, convert it to LONG IF.

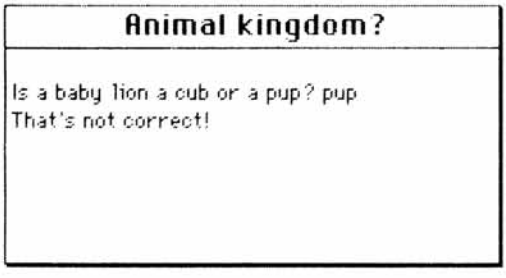
Exercise 8.1

Type in the program below, run it, and fill in what is missing in the window:

```
WINDOW 1,"Mr or Ms?",(0,0)-(210,80)

TEXT _geneva, 9
PRINT

INPUT "What is your last name? : ";you$
INPUT "Male or female (m or f)?: ";sex$

LONG IF sex$ ="m"
  PRINT "Hello Mr. "you$
XELSE
  PRINT "Hello Ms. "you$
END IF

STOP
```

Exercise 8.3

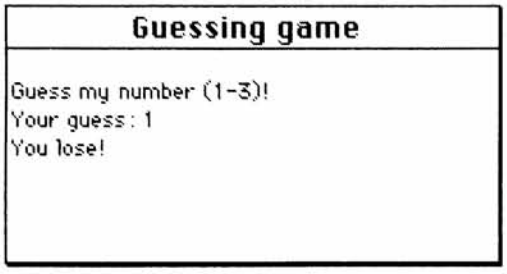
Look at the window and fill in the missing lines in the program below:




```
WINDOW 1,"Animal kingdom?",(0,0)-(210,90)

TEXT _geneva, 9
PRINT

_____________________________________

LONG IF ans$ = "cub"

XELSE

_____________________________________

END IF

STOP
```

Exercise 8.4

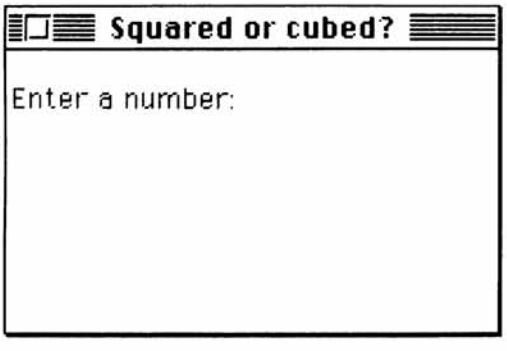
Look at the window and fill in the missing lines in the program below:




```
WINDOW 1,"Guessing game",(0,0)-(210,90)

TEXT _geneva, 9
PRINT

_____________________________________________

_____________________________________________

LONG IF ans = 2
  PRINT "••• You Win!!! •••"
  BEEP
  BEEP
XELSE
_____________________________________________
END IF

STOP
```

Exercise 8.6

Type in the program below, run it, and fill in what is missing in the window:

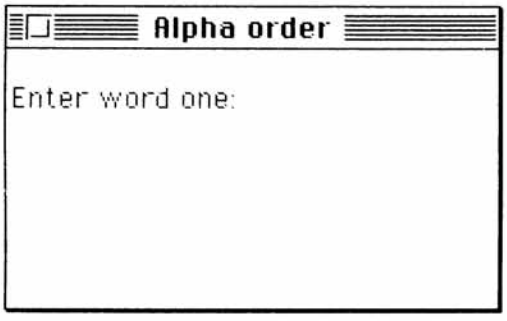



```
WINDOW 1,"Squared or cubed?",(0,0)-(210,120)

PRINT
INPUT "Enter a number: ";num!
INPUT "Squared or cubed? ";ans$

LONG IF UCASE$(ans$) = "SQUARED"
  PRINT "Squared="num!*num!
END IF

LONG IF UCASE$(ans$) = "CUBED"
  PRINT "Cubed = "num!*num!*num!
END IF

STOP
```

UCASE$ is used to return all the characters of a string as capital letters. This allows you to compare one time instead of for every possibility of case the user may have entered.

Exercise 8.8

Type in the program below, run it, and fill in what is missing in the window:




```
WINDOW 1,"Alpha order",(0,0)-(210,110)

PRINT

INPUT "Enter word one: ";word1$
INPUT "Enter word two: ";word2$

PRINT "Alphabetically speaking"

LONG IF UCASE$(word1$) > UCASE$(word2$)
  PRINT word2$ " comes first"
XELSE
  PRINT word1$ " comes first"
END IF

STOP
```

Strings can be compared this way because character strings are numbers to the computer. The letter "A" is a 65. The letter "G" is a 71. When comparing two characters, 65 is less than 71. This will be discussed more in Lesson 13.

Programming problems

Program problem 8.1

Write a program that asks for a number. Have the program print whether the number is more than, less than or equal to 100.

Program problem 8.2

Write a program that asks for two names. Tell the user which name comes first alphabetically.

Lesson 9

Repeating program lines (looping)

This lesson will discuss DO-UNTIL and WHILE-WEND loops.

Since these statements repeat a series of program lines enclosed within them over and over again, they are referred to as loops. When a certain condition is met, the loop will exit to the next line following the end of loop statement.

For example, if a program is calculating payroll for many employees, it can use the same program lines to calculate the pay for each employee. When the last employee is completed, the loop is exited. Being able to re-use program lines saves programming time.

The UNTIL and WHILE statements use compare symbols, like those used with IF and LONG IF, to determine if it's time to exit a loop.

The difference between the two types of loops is:

- DO/UNTIL does all the program lines between DO and UNTIL at least once, then if the condition at the UNTIL statement is false, it repeats the lines. If the condition at UNTIL is true, the loop exits to the line following the UNTIL statement.
- WHILE/WEND repeats program lines within the loop only if the condition is true. Otherwise, program lines are skipped and the loop exits to the line after WEND.

These two loops are laid out like this:

```
DO
   - program lines -
   - program lines -
UNTIL something = _true


WHILE something = _true
   - program lines -
   - program lines -
WEND
```

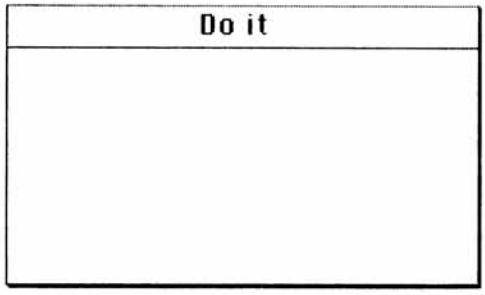
If the variable *something* is _false, the program lines between DO and UNTIL will be executed. Program lines will *always* be executed at least once.

As long as the variable *something* is _true, program lines between WHILE and WEND will be executed. If *something* is not _true, program lines will not be executed at all.

Besides checking the condition at the end of the loop, the main difference between the DO loop and the WHILE loop is that DO will always be executed at least once.

Exercise 9.1

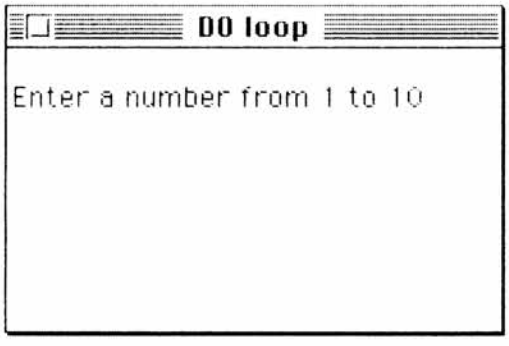
Type in the program below, run it, and fill in what is missing in the window:




```
WINDOW 1,"Do it",(0,0)-(210,100)

counter = 0

DO
  counter = counter + 1
  PRINT counter;
UNTIL counter >= 26

STOP
```

Exercise 9.2

Type in the program below, run it, and fill in what is missing in the window:



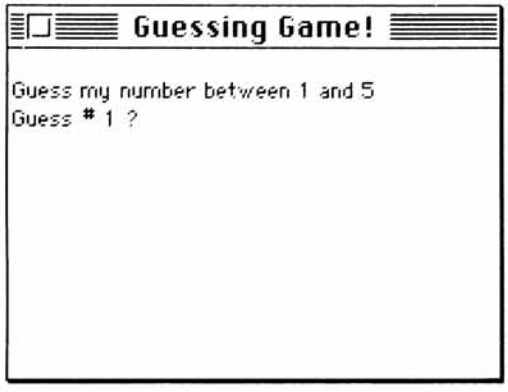

```
WINDOW 1,"DO loop",(0,0)-(210,120)

PRINT
INPUT "Enter a number from 0 to 10 :";num

counter = 0

DO
  counter = counter + 1
  PRINT "Hello"counter;
UNTIL counter >= num

STOP
```

Note that even when zero is entered, the DO *loop is still executed at least once.*

Exercise 9.3

Type in the program below, run it, and fill in what is missing in the window:

```
WINDOW 1,"Guessing Game!",(0,0)-(210,140)
TEXT _geneva, 9

myGuess = RND(5)          'get rnd num 1-5
trys    = 0


PRINT
PRINT "Guess my number between 1 and 5"

DO
  trys = trys + 1
  PRINT "Guess #"trys;
  INPUT guess             'get user's guess
UNTIL guess = myGuess OR trys > 5

LONG IF guess = myGuess
  PRINT "You guessed my number! ("guess")"
  PRINT "It took you"trys"try";
  IF guess > 1 THEN PRINT "s"
XELSE
  PRINT "You lose!"    'took six trys!
END IF

STOP
```



This RND function returns a random number between one and five and puts it into the variable *myGuess*. Random is discussed more in Lesson 10.

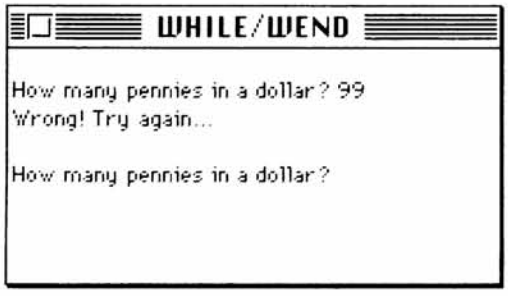
OR allows you to compare more than one value on the same line. This line would be true if either trys is greater than five or guess equals myGuess. If AND were used instead, this would be true only if both compares is true.

The OR and AND statements can be used with comparison signs in DO/UNTIL, WHILE/WEND, IF/THEN, LONG IF, and other comparison statements. See the *Reference* manual for more information.

Exercise 9.4

Type in the program below, run it, and fill in what is missing in the window:

```
WINDOW 1,"WHILE/WEND",(0,0)-(210,100)
TEXT _geneva, 9

answer = _false
trys   = 0

WHILE answer = _false OR trys <= 3
  PRINT
  INPUT "How many pennies in a dollar? ";num

  LONG IF num = 100
    answer = _true
    PRINT "Good thinking, Einstein!"
  XELSE
    PRINT "Wrong! Try again..."
    trys = trys + 1
  END IF

WEND

STOP
```



The constants `_true` and `_false` are used to make programs easier to read. As long as the answer is equal to `_false`, the program lines in the loop will be repeated.

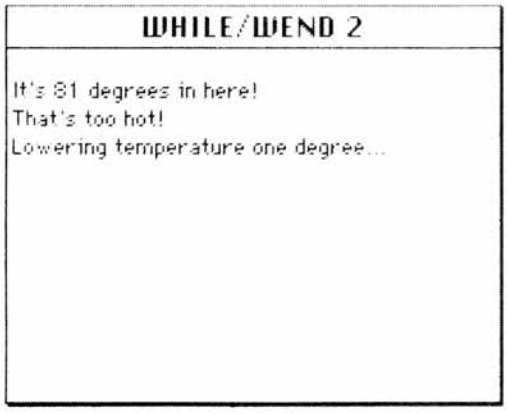
You should always provide a way for confused users to get out of the loop.

A `WHILE` loop checks the condition at the beginning of the loop. If the condition is not true, the loop exits at the line following the `WEND` statement.

Exercise 9.5

Type in the program below, run it, and fill in what is missing in the window:

```
WINDOW 1,"WHILE/WEND 2",(0,0)-(210,150)
TEXT _geneva, 9

temperature = 81

WHILE temperature > 79
  PRINT
  PRINT "It's"temperature"degrees in here!"
  PRINT "That's too hot!"
  PRINT "Lowering temperature one degree..."
  temperature = temperature - 1
  DELAY _sec
  TRON x              'cmd-period exit
WEND

PRINT
PRINT "Ahhh! That's better!"

STOP
```



The DELAY statement makes the computer pause for a specified amount of time.

TRON x, which stands for "trace-on", scans the keyboard to see if the ⌘-period keys are being held down. STOP is executed if the keys are being pressed.

Read the first page of the MacsBug chapter of the Handbook for a way to escape endless loops gracefully.

Exercise 9.6

Modify the program below so it ends at 0 and is not stuck in an endless loop:

```
WINDOW 1,"WHILE/WEND 2",(0,0)-(210,150)
TEXT _geneva, 9
counter = 20
WHILE counter > 0
  PRINT counter;
  counter = counter + 1
  TRON x
WEND

STOP
```

Understanding endless loops and how to avoid them is an important part of debugging.

After you fix the endless loop, remove this line from the program. It should only be used for testing purposes.

Exercise 9.7

Modify the program below so it is not an endless loop:

```
WINDOW 1,"ENDLESS LOOP 2",(0,0)-(210,150)
TEXT _geneva, 9
counter = 20
DO
  PRINT counter;
  counter = counter + 1
  TRON x
UNTIL count > 100
STOP
```

As discussed in a previous lesson, this type of bug is very common and should be one of the first you look for when your program stops functioning.

Exercise 9.8

Run this program and explain what happens:

```
WINDOW 1,"Key wait!",(0,0)-(210,150)
TEXT _geneva, 9

PRINT "Press a key or click"
PRINT "the mouse button to stop!"

DO
UNTIL  LEN(INKEY$) OR FN BUTTON
```

`LEN(INKEY$)` will be true if a key is pressed.

`FN BUTTON` will be true if the mouse button is held down.

Programming problems

Program problem 9.1

Write a program that asks the question: "What is the sum of 2+2?". If the user types in the wrong answer, print a response like "You made a mistake!" and ask the question again until they get the answer correct or exceed 4 tries. If they get it correct, print "Very good, you got it correct!". Print the number of tries it took to guess the answer.

Lesson 10

Numeric functions

A function returns a value of some kind. Numeric functions return a floating point or integer value.

These four numeric functions are are very useful and simple.

`ABS(n)` Returns the absolute value of *n* (distance from zero).

`RND(n%)` Returns a random integer number in the range of 1 to *n%* (maximum of 32,767).

`VAL(n$)` Returns the numeric value of the first number in a string (ignores spaces but not characters).

`INC(n)` Returns the value of n plus one. Functionally equivalent to n = n + 1. Long and short integers only.

`USING` Formats the way numbers are displayed. For example, one way to format dollar values is to do this:

```
PRINT USING "$#,###.##"3223.3472
```

Program output:

```
$3,223.35
```

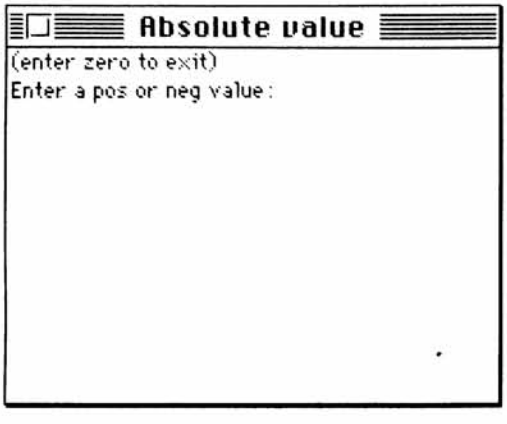
You may be asking yourself "Why do I need to know these functions?"

Since computer programs use numbers so often, certain functions, like these, are used all the time in all sorts of routines. Having some knowledge of them will help you implement your programming ideas.

Exercise 10.1

The absolute value of a number is how far the number is from zero. Type in the program below, run it and enter what is missing in the window:

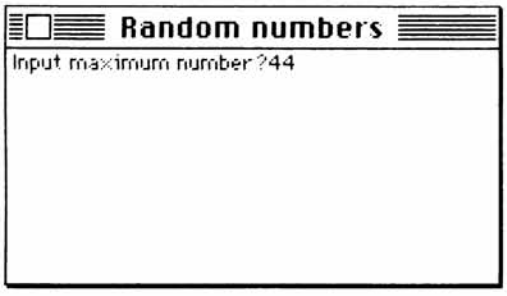
```
WINDOW 1,"Absolute value",(0,0)-(210,150)
TEXT _geneva, 9

DO
  PRINT "(enter zero to exit)"
  INPUT "Enter a pos or neg value: ";val!
  PRINT "The absolute value is "ABS(val!)
UNTIL val! = 0

DELAY 2 * _sec
```



This allows you to test as many values you want. When you enter zero, the loop will be exited here. After a two second delay, the program will quit.

Exercise 10.2

Type in the program below and enter what is missing in the window:

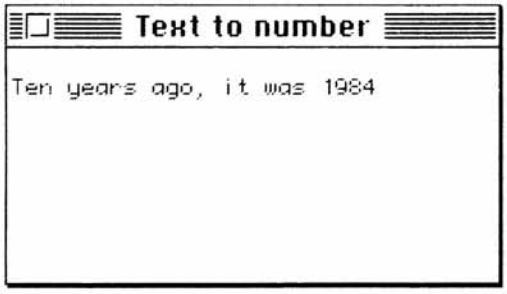



```
WINDOW 1,"Random numbers",(0,0)-(210,100)
TEXT _geneva, 9

INPUT "Input maximum number? "; maxNum
counter = 1          'initialize variable
   .
PRINT
PRINT "Printing ten random"
PRINT "numbers between 1 and" maxNum
PRINT

DO
  counter = counter + 1
  PRINT RND(maxNum);
UNTIL counter > 10

DO
UNTIL LEN(INKEY$) OR FN BUTTON
```

Exercise 10.3

Type in the program below and enter what is missing in the window:

```
WINDOW 1,"Text to number",(0,0)-(210,100)
TEXT _monaco, 9

theYear% = 1900+VAL(RIGHT$(DATE$,2))
PRINT
PRINT"Ten years ago, it was";theYear%-10

DO
UNTIL LEN(INKEY$) OR FN BUTTON
```

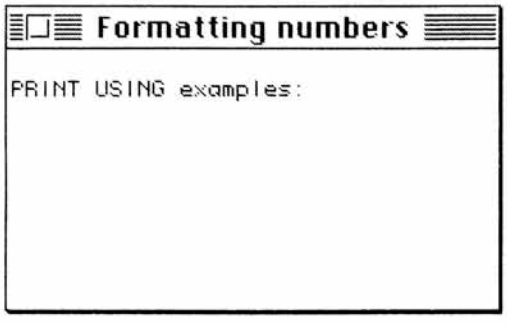


The DATE$ function returns the date in the format "12/01/93" (explained in Lesson 13).

Exercise 10.4

Type in the program below, run it, and enter what is missing in the window:

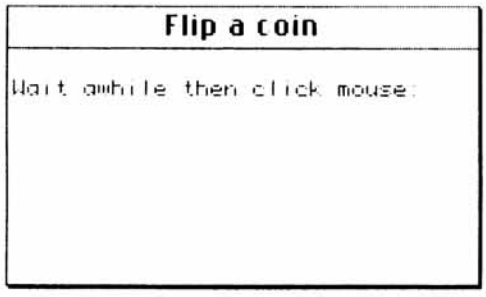
```
WINDOW 1,"Formatting numbers",(0,0)-(210,110)
TEXT _monaco, 9

PRINT
PRINT "PRINT USING examples:"

PRINT "1/3 dollar = "1/3
PRINT "1/3 dollar = ";USING"$###.##";1/3.

PRINT
PRINT "1/10,000 = ";1/10000
PRINT "1/10,000 = ";USING ".######";1/10000

DO
UNTIL LEN(INKEY$) OR FN BUTTON
```



"1E-04" is called scientific notation. It describes very large numbers. 1E-04 is equal to .0001. See Scientific notation in the Math appendix of the Reference manual.

Exercise 10.5

Type in the program below and enter what is missing in the window:



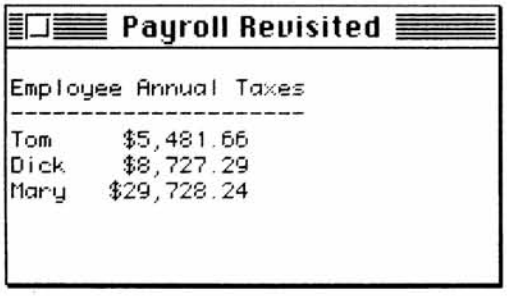

```
WINDOW 1,"Flip a coin",(0,0)-(200,100)
TEXT _monaco, 9          'set font

head& = 0                'head count
tail& = 0                'tail count
  .
PRINT
PRINT "Wait awhile then click mouse:"

DO                       'flip coins
  LONG IF RND(2) = 1
    INC(head&)
  XELSE
    INC(tail&)
  END IF
UNTIL FN BUTTON

PRINT
PRINT "Heads came up "head&"times
PRINT "Tails came up "tail&"times
PRINT
PRINT "Press a key to quit."

DO
UNTIL LEN(INKEY$)
```

These two long integer variables are used to hold the number of times heads and tails are flipped. Long integer variables are used because values easily exceed the 32,767 limit of short integers.

This is equivalent to: `head& = head& + 1`

Exercise 10.6

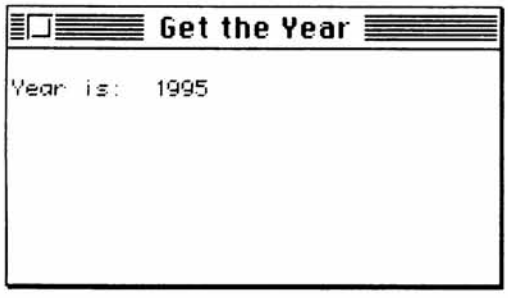
Look at the window and fill in the missing lines in the program below:




```
WINDOW 1,"Payroll Revisited",(0,0)-(210,100)
TEXT _monaco, 9

tom!  = 23833.33
tTax! = .23

dick! = 32323.32
dTax! = .27

mary! = 78232.23
mTax! = .38

PRINT: PRINT "Employee Annual Taxes"
PRINT "---------------------"
PRINT "Tom  "USING "$##,###.##";tom! * tTax!

_____________________________________________


DO
UNTIL LEN(INKEY$) OR FN BUTTON
```

Exercise 10.7

Look at the window and fill in the missing line in the program below:

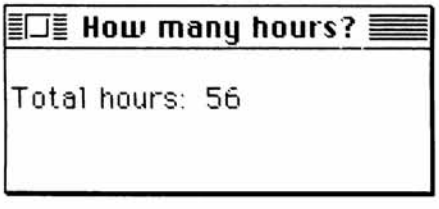



```
WINDOW 1,"Get the Year",(0,0)-(210,100)
TEXT _monaco, 9

tDate$ = "12/02/95"

PRINT '
```

```
DO
UNTIL LEN(INKEY$) OR FN BUTTON
```

Exercise 10.8

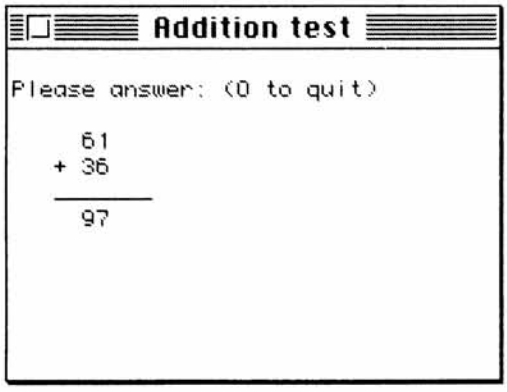
Look at the window and fill in the missing lines in the program below:



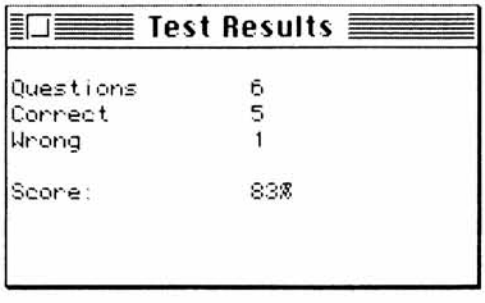

```
WINDOW 1,"How many hours?",(0,0)-(180,60)
hrs1$ = "23 hours"
hrs2$ = "33 hours"
```

Exercise 10.9

Look at the two windows on the right and fill in the missing lines in the program below:



This program gets two random numbers and asks for the answer to the sum of them. Enter zero to finish the program and get the results which are printed in the window below:




```
WINDOW 1,"Addition test",(0,0)-(210,140)
TEXT _monaco, 9

yes = 0
no  = 0

DO
  num1 = RND(100)
  _____________________________________________
  CLS
  PRINT
  PRINT"Please answer: (0 to quit)"
  PRINT
  _____________________________________________
  PRINT "    +" num2
  PRINT "    _______"
  INPUT "      ";answer
  PRINT

  LONG IF answer = num1 + num2
    PRINT "    Correct!"
    yes = yes + 1
  XELSE
    LONG IF answer >< 0
      BEEP
      PRINT "    Incorrect!"
      no = no + 1
    END IF
  END IF
  DELAY _sec
UNTIL answer = 0 OR yes + no > 5

WINDOW #2,"Test Results",(0,0)-(200,100)
TEXT _monaco, 9
total = yes + no
PRINT
_____________________________________________
PRINT
PRINT "Score: ",USING "###%";yes/total*100
DELAY _sec

DO
UNTIL LEN(INKEY$) OR FN BUTTON
```

Programming problems

Program problem 10.1

Modify *Exercise 10.9* to do subtraction problems.

Program problem 10.2

Modify *Exercise 10.9* to do multiplication problems.

Program problem 10.3

Write a program that plots random dots all over the window. Sort of like "Twinkle-Twinkle Little Star".

Program problem 10.4

Write a program that plots random lines in the window useing random line patterns. If you have a color Mac, try random colors too. Hint: See the `LONG COLOR` statement in the *Reference* manual and the *Color* chapter of the *Handbook.*

Lesson 11

Counter loop

FOR/NEXT loop statements can be used to execute a series of statements a specific number of times. For example, the FOR/NEXT loop below will execute the PRINT statement, between FOR and NEXT, twently times (while it counts from one to twenty).

```
FOR count = 1 TO 20
  PRINT count;
NEXT count
```

FOR/NEXT loops can count incrementally. For example, the following loop will print the numbers 5, 10, 15, 20 and 25:

```
FOR myVar = 5 TO 25 STEP 5
  PRINT myVar;
NEXT myVar
```

The following FOR/NEXT loop will print numbers counting backwards, from 20 to 1:

```
_backwards = -1      'define a constant

FOR counter = 20 TO 1 STEP _backwards
  PRINT counter;
NEXT
```

Putting the variable after NEXT is optional.

Program lines inside a FOR/NEXT loop are indented the same way program lines are indented in other loop structures.

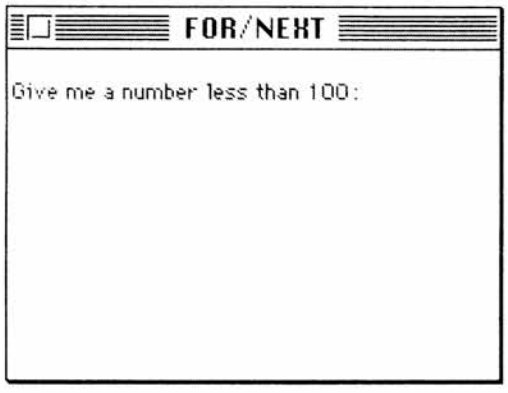
As with other loops, you should be careful to avoid endless loops. To insure that you don't get "stuck" in an endless loop, use TRON x on one of the lines between the FOR and NEXT lines when you are testing a loop.

After the loop is working as you expect, remove the line containing TRON x. This will speed the loop up.

Exercise 11.1

Type in the program below, run it, and fill in what is missing in the window:

```
WINDOW 1,"FOR/NEXT",(0,0)-(210,140)
TEXT _geneva, 9

PRINT
INPUT "Give me a number less than 100: ";pt%
   .
LONG IF pt% > 0 AND pt% < 100
  PRINT "It's easy to count forward to" prt%

  FOR count = 1 TO pt%
    PRINT count;
  NEXT count

END IF

DO
UNTIL LEN(INKEY$) OR FN BUTTON
```



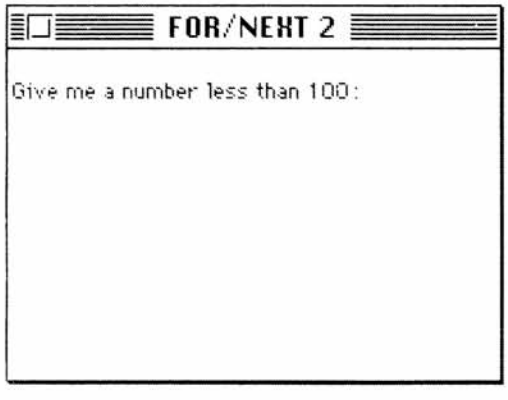
FOR/NEXT loops are good for repeating part of a program a specfied number of times. DO and WHILE loops are perfect for repeating part of a program until a certain condition is met.

Loops appearing inside other loops or structures must be completely inside. You can't have overlapping loops.

Exercise 11.2

Type in the program below, run it, and fill in what is missing in the window:

```
WINDOW 1,"FOR/NEXT 2",(0,0)-(210,140)
TEXT _geneva, 9: PRINT

INPUT "Give me a number less than 100: ";prt%

LONG IF prt% > 0 AND prt% < 100'in range?
  PRINT "Count backward from"prt%"to one!"
  FOR count = prt%  TO 1 STEP -1
    PRINT count;
  NEXT
END IF

STOP
```

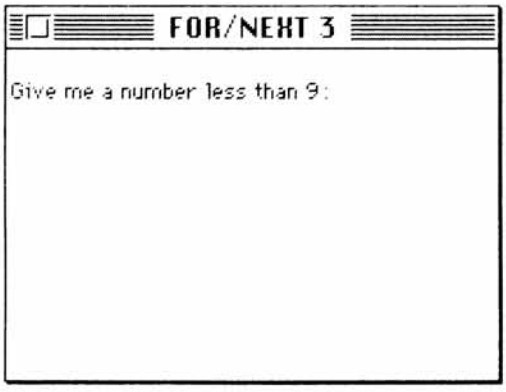


A negative step makes the loop count backwards.

Exercise 11.3

Type in the program below, run it, and fill in what is missing in the window:

```
WINDOW 1,"FOR/NEXT 3",(0,0)-(210,140)
TEXT _geneva, 9

PRINT
INPUT "Give me a number less than 9: ";prt!

LONG IF prt! > 0 AND prt! < 9 'in range?
  PRINT "I can count in fractions too!"
  FOR count! = 0 TO prt! STEP .25
    PRINT count!;
  NEXT count!
END IF

STOP
```

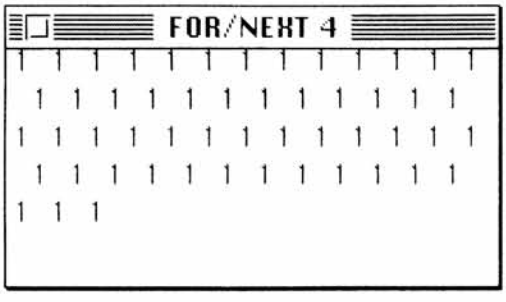


Be sure "count!" and "prt!" are floating point to avoid an endless loop.

Exercise 11.4

Endless loops can happen whenever the counter variable never gets set to final value. Type in the program below, run it, and fill in what is missing in the window:

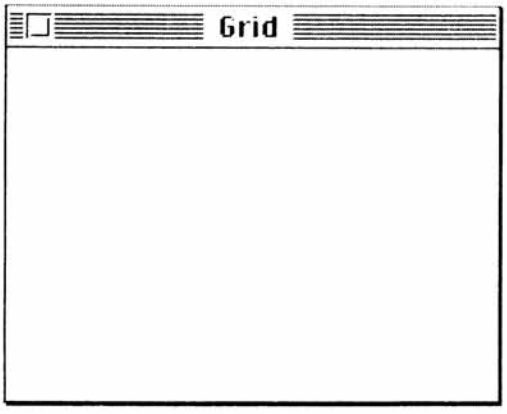
```
WINDOW 1,"FOR/NEXT 4",(0,0)-(210,100)

FOR count = 1 TO 5 STEP .25
  PRINT count;
  TRON x
NEXT

STOP
```



Use this to enable the command-period keys. This loop is endless because the integer variable, *count*, cannot hold a floating point value. With integers, when .25 is added to 1, the result is still one. Change both occurances of *count* to *count!* to fix the problem so the program runs properly.

Exercise 11.5

Look at program and fill in what is missing in the window:




```
WINDOW 1,"Grid",(0,0)-(210,150),_docNoGrow

FOR h = 0 TO 220 STEP 10

  FOR v = 0 TO 140 STEP 10
    PLOT 0,v TO 220,v
    PLOT h,0 TO h,150
  NEXT

NEXT

DO
UNTIL FN BUTTON OR LEN(INKEY$)
```

Programming problems

Program problem 11.1

Write a program that prints the years 1980 through 1995 and prints a random money amount greater than a million and less than 10 million to the right of the year. Make sure the money values are formatted like this: $34,444,567.

Program problem 11.2

Write a program that calculates and prints the result of the numbers 1 through 7 divided by 33. Print the numbers formatted neatly.

Lesson 12

Functions

Functions are used to separate logical parts of a program into smaller, easier to understand segments. This makes programs easier to read, maintain and debug.

LOCAL FN is used to make functions. The basic types of local functions are shown below.

The most basic form of a function looks like this:

```
LOCAL FN doGreeting
  PRINT "Hello"
END FN

FN doGreeting
PRINT "Goodbye"
```

This line calls the function above it with the same name. After the function prints "Hello", END FN sends the program back to the statement following the one that called it, which prints "Goodbye" .

Another type of function is one that is passed values. One or more values can be passed to a list of parameters enclosed in parentheses. For example:

```
LOCAL FN printStrings (num%, strng$)
  WHILE num% > 0
    PRINT strng$
    num% = num% - 1
  WEND
END FN

FN printStrings (13, "hello there")
PRINT "bye-bye"
```

This line calls the function above it with the same name. The function is passed the value 13 in *num%*, and "hello there" in *strng$*. After *"hello there"* is printed thirteen times, END FN sends the program back to the statement following the one that called it, which prints "bye-bye" .

The reason you need to pass values to a function is because functions are like islands. They cannot see or change variables in other functions, and other functions cannot see or change variables in that function. A later lesson will show you how to define global variables that can be seen and changed by all functions.

The last type of function can pass a value back to the statement calling it. For example:

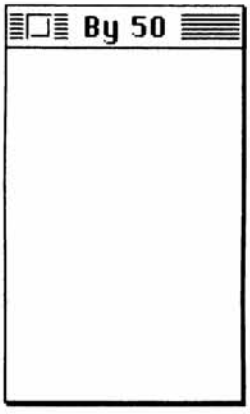
```
LOCAL FN timesTen(myVar)
  myVar = myVar * 10
END FN = myVar

num% = FN timesTen(50)
PRINT num%
```

This line calls the function above it with the same name, passing 50 to the variable *myVar*. After *myVar* is multiplied by ten, END FN passes the value of *myVar* to the variable *num%*, then sends the program back to the statement following the one that called it,which prints 500.

Exercise 12.1

Type in the program below, run it, and fill in what is missing in the window:

```
'-------- Functions ------------
LOCAL FN initializeWindow
  WINDOW 1,"By 50",(0,0)-(100,150)
  TEXT _monaco, 9
  PRINT
END FN

LOCAL FN doForLoop
  FOR x = 50 TO 500 STEP 50
    PRINT x
  NEXT
END FN

'--------- Main program ---------
FN initializeWindow
FN doForLoop

DO   'key or mouse click to quit
UNTIL FN BUTTON OR LEN(INKEY$)
```



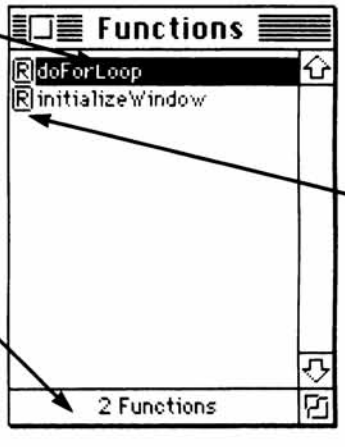
Notice the layout of this program. Functions are defined at the top of the program because only lines below a `LOCAL FN` can call them. Headers, like *Functions* and *Main program*, make it easier to find parts of a program. Headers also make programs easier to read and debug.

This is the first line of the program! Lines inside a `LOCAL FN` are ignored until they are called from a line like this. This is the first line of this program that is executed.

Look at functions being called from the *Main* section of a program to get a quick idea of what the program is doing.

To make the editor find and display a particular function, simply click on the function in this window.

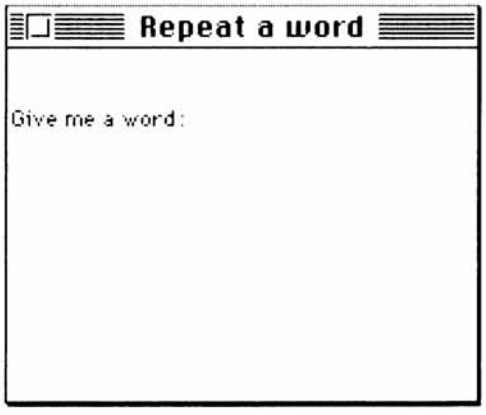
Click the bottom of this window to display the *functions* view if it isn't showing already. See the *Tool menu* section of the *Reference* manual for more information about this window.



Click on the circled-R to find the first reference to the function. Click again to find the next reference, and so on, until you hear a beep signifying that there are no more references to that function.

Exercise 12.2

Type in the program below, run it, and fill in what is missing in the window:



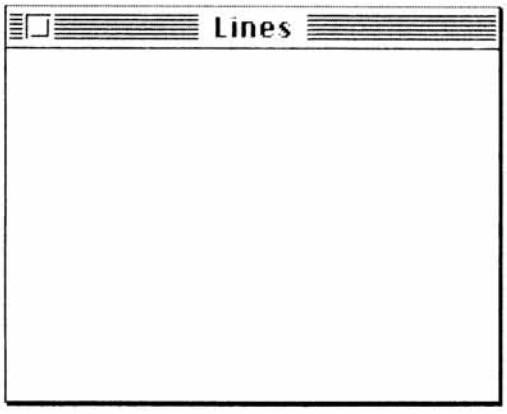

```
'--------- Functions --------
LOCAL FN makeWindow
  WINDOW 1,"Repeat a word",(0,0)-(210,150)
  TEXT _geneva, 9
END FN

LOCAL FN setUpWindow
  CLS
  PRINT
  PRINT
END FN

LOCAL FN printWord (howMany, wordToPrint$)
  FN setUpWindow

  FOR x = 1 TO howMany
    PRINT wordToPrint$", ";
  NEXT
  DELAY _sec * 2              '2 sec delay
END FN

LOCAL FN multiWords
  FN setUpWindow

  INPUT "Give me a word: ";yourWord$
  INPUT "Number 1 to 100 (0 to quit): ";num
  IF NUM THEN FN printWord(num,yourWord$)
END FN = num

'--------- Main program ---------
FN makeWindow

DO
  quitTime = FN multiWords
UNTIL quitTime = 0
```

Following the logic of a program with local functions is done by following the function names in the Main program. For example, to trace the logic of this program, look at the functions as described below:

1. *Calls "makeWindow" which opens a window and sets text.*
2. *Calls "multiWords" each time through the loop, which calls the "setUpWindow" and "printWord" functions, causing the user's text to be printed so many times. The "printWord" function also calls the "setUpWindow" function.*

Exercise 12.3

Type in the program below, run it, and fill in what is missing in the window:

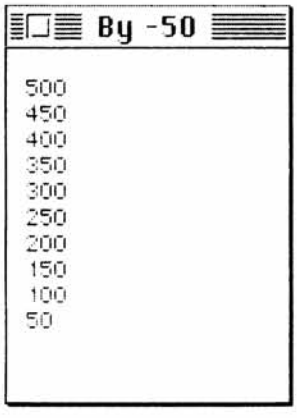



```
'--------- Functions --------
LOCAL FN initializeWindow
  WINDOW 1,"Lines",(0,0)-(210,150),_docNoGrow
  TEXT _geneva, 9
END FN

LOCAL FN doLines
  COLOR RND(8)
  PLOT TO RND(230), RND(150)
END FN

'--------- Main program ---------
FN initializeWindow

DO
  FN doLines
UNTIL FN BUTTON OR LEN(INKEY$)
```

Don't add this color line if you have a black and white monitor. Add a line that changes the lines to random patterns instead.

Exercise 12.4

Look at the window and fill in the missing lines in the program below.

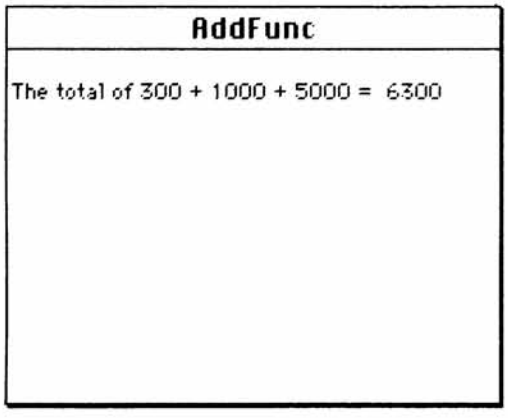



```
'--------- Functions --------
_____________________________________

'--------- Main program ---------
```

Exercise 12.5

Type in the program below, run it, and fill in what is missing in the window:

```
'--------- Functions --------
LOCAL FN initializeWindow
  WINDOW 1,"AddFunc",(0,0)-(210,150),_docNoGrow
  TEXT _geneva, 9
END FN

LOCAL FN addNumbers&(num1, num2, num3)
  result& = num1 + num2 + num3
END FN = result&

LOCAL FN doExample
  PRINT
  PRINT "The total of 300 + 1000 + 5000 = ";
  PRINT FN addNumbers&(300, 1000, 5000)

  val1 = 300
  val2 = 1000
  val3 = 5000
  PRINT "The total of"val1"+"val2"+"val3"= ";
  PRINT FN addNumbers&(val1, val2, val3)

  try3 = 300
  _val = 5000
  PRINT "The total of "val1"+ 1000 +"_val"= ";
  PRINT FN addNumbers&(try3, 1000, _val)

  try4 = 100
  PRINT "The total of "try4*3"+ 1000 +"_val"= ";
  PRINT FN addNumbers&(try4*3, 1000, _val)
END FN

'--------- Main program ---------
FN initializeWindow
FN doExample

STOP
```



This function is passed three numbers that it adds together. It then passes back the result to the calling routine as a long integer value.

This function calls the *addNumbers&* function several ways to show how it can be passed values as numbers, constants, variables or expressions.

You can print the value of a function directly using this method. You can also include a function in an expression. For example, the following expression:

```
ans& = FN addNumbers&(5,10,15) * 1000
```

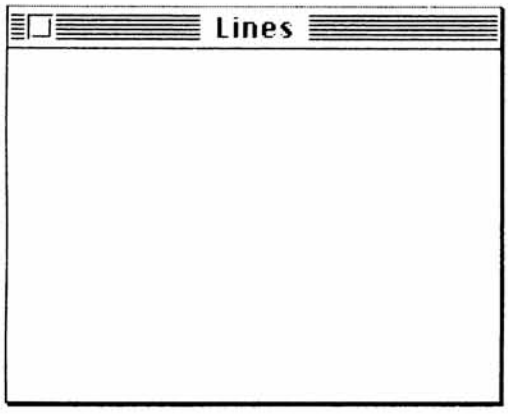
would result in the variable *ans&* being assigned the value of 30,000.

Follow the functions being called in the Main program to understand what the program is doing:

1. *initializeWindow*
   open window
   set text Geneva 9
2. *doExample*
   print numbers being added
   print val returned by addNumbers

Exercise 12.6

Type in the program below, run it, and fill in what is missing in the window:




```
'--------- Functions --------
LOCAL FN initializeWindow
  WINDOW 1,"Function$",(0,0)-(160,70)
  TEXT _geneva, 12
END FN
   .
LOCAL FN addStrings$ (a$, b$)
  c$ = a$ + b$
END FN = c$

LOCAL FN doExample
  d$ = FN addStrings$("Hi", " there!")
  PRINT
  PRINT d$
END FN

'--------- Main program ---------
FN initializeWindow
FN doExample

STOP
```

If a function is returning a value, its name must have a symbol after it describing the type of data being returned---a dollar sign representing the string value being returned in this case.

If the dollar sign ($) symbol was not used in this example, an error would result.

Programming problems

Program problem 12.1

Convert *Exercise 10.6* to a program using local functions.

Program problem 12.2

Convert *Exercise 10.9* to a program using local functions.

Program problem 12.3

Write a program that has a function that is passed three integer parameters: a, b, and c, and returns their average.

Lesson 13

Text functions

This lesson covers a number of text functions you will find indispensable.

`STR$` Returns a text string equal to a number or numeric variable. This is the opposite of the `VAL` function described in lesson 10.

`LEN` Returns the total number of characters contained in a text string.

`UCASE$` Returns a text string with all characters converted to capital letters (uppercase).

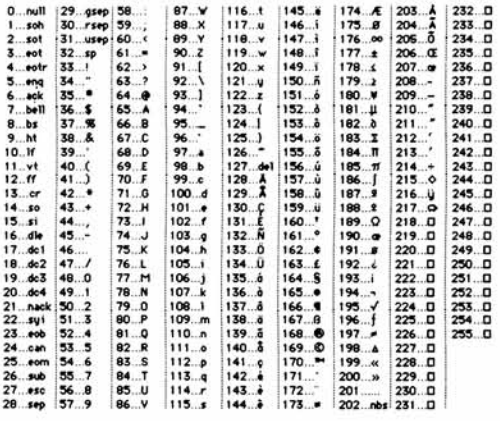
`INKEY$` Get a one character string equal to the key pressed on the keyboard. This will not wait for a key press but will return the character if the key was pressed.

`TIME$` Returns the time as a string in the format hh:mm:ss (hours, minutes, seconds). If it was 7:13 pm, this would return: 18:13:00

`DATE$` Returns the date as a string in the format: mm/dd/yy (month, day, year). If today was January, 23, 1995, this would return the string: 01/23/95

`ASC` Returns the ASCII number code of the first character in a text string. This will be a value in the range of zero through 255. For example, `ASC("A")` would return the number 65. ASCII is an acronym for *American Standard Code for Information Interchange*.

`CHR$(n)` Returns a single character string with the ASCII value of *n*. For example, the statement: `PRINT CHR$(65)` would print the letter "A".



ASCII codes used with `CHR$` and `ASC` (Geneva).

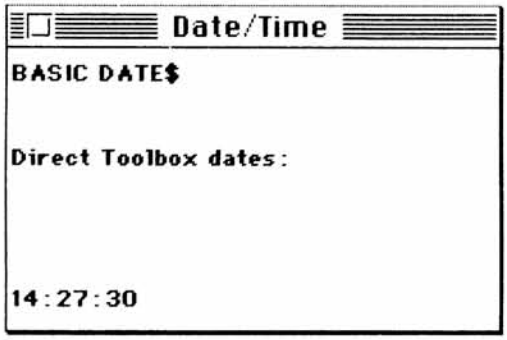
Note: While characters are generally the same for codes 32 through 122, other character codes may have different characters in different fonts. You can see a bigger chart under `CHR$` in the *Reference* manual.

You may be asking yourself "Why do I need to know these string functions?"

Since computer programs must decipher input from human beings, who, unlike computers, greatly dislike communicating with numbers, functions like these for manipulating characters, words and text are invaluable.

Exercise 13.1

Type in the program below, run it, and fill in what is missing in the window:




```
'--------- Functions --------
LOCAL FN buildWindow
  WINDOW 1,"Infotime",(0,0)-(210,130)
  TEXT _geneva, 9
END FN

LOCAL FN showDate
  TEXT  ,,_boldBit%
  PRINT %(2,14);
  PRINT "BASIC DATE$"
  TEXT  ,, 0
  PRINT DATE$;"  (mm:dd:yy)"
END FN

LOCAL FN showToolboxDates
  CALL IUDATESTRING([_time],_shortDate, SD$)
  CALL IUDATESTRING([_time],_abbrevDate,AD$)
  CALL IUDATESTRING([_time],_longDate,  LD$)
  TEXT ,, _boldBit%
  PRINT : PRINT "Direct Toolbox dates:"
  TEXT  _monaco,9, 0
  PRINT "Short: "SD$
  PRINT "Abrev: "AD$
  PRINT "Long : "LD$;
END FN

LOCAL FN updateTime
  COLOR _zRed : TEXT  ,,_boldBit%
  PRINT %(0,110);
  CLS LINE
  PRINT %(0,110);TIME$;
  DELAY _secHalf
END FN

'--------- Main program ---------
FN buildWindow
FN showDate
FN showToolboxDates

DO
  FN updateTime
UNTIL FN BUTTON OR LEN(INKEY$)
```

This function uses FutureBASIC's DATE$ function to display the simple date.

This function uses three toolbox calls to get the date in other formats. These are described in detail in *Inside Macintosh*, *Processes: Time Manager section.*

These are brackets. Their keys are above the *return* key.

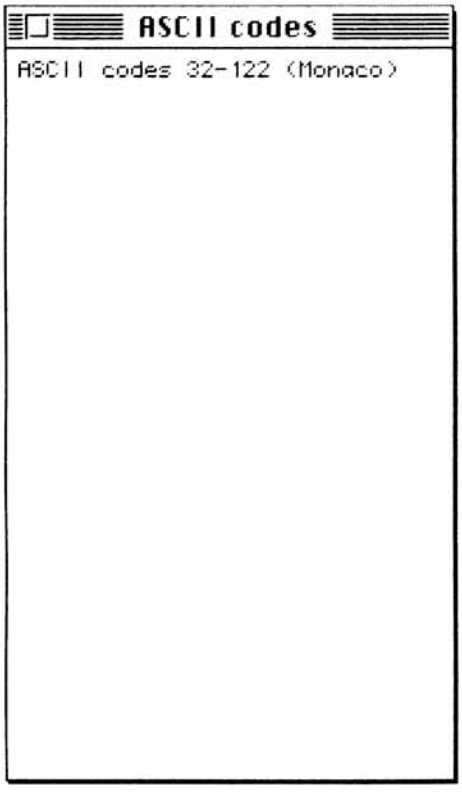
This function is called each time through the main program loop to update the time.

This half-second delay is used to eliminate screen flash. This means that the time is printed only every half second, not continuously.

Exercise 13.2

Type in the program below, run it, and fill in what is missing in the window:

```
'--------- Sub-programs --------
LOCAL FN buildWindow
  WINDOW 1,"ASCII codes",(0,0)-(190,310)
  TEXT _monaco, 9
END FN

LOCAL FN displayASCIICharacters

  PRINT%(5,12) "ASCII codes 32-122 (Monaco)";

  FOR x = 32 TO 96 STEP 32
    FOR y = 0 TO 31
      LONG IF x + y <= 122
        PRINT%((x-32)*1.8, y*9+25);
        PRINT USING "###"; y+x;
        PRINT " = " CHR$(x+y);
        DELAY 100
      END IF
    NEXT
  NEXT
END FN

'--------- Main program ---------
FN buildWindow
FN displayASCIICharacters

DO
UNTIL FN BUTTON OR LEN(INKEY$)
```



This line calculates where the text will be printed and puts the cursor there. The semicolon means the next line will print at that spot.

Formats and prints the ASCII numbers so they line up. The semicolon means the next line will print immediately after this.

Prints an equal sign and the ASCII character. The semicolon stops scrolling of text printed near the bottom of the window.

The `DELAY 100` statement makes text print in slow motion to help illustrate how text position is calculated. It isn't necessary otherwise.

Exercise 13.3

Type in the program below, run it, and fill in what is missing in the window:

```
'--------- Functions --------
LOCAL FN initializeWindow
  WINDOW 1,"Length",(0,0)-(210,220)
  TEXT _geneva, 9
END FN

LOCAL FN doWord(txt$)
  CLS
  PRINT
  PRINT "The ASCII code of the first"
  PRINT "character, ""CHR$(ASC(txt$));
  PRINT """, is:"ASC(txt$)
  PRINT
  PRINT "There are "LEN(txt$)" characters"
  PRINT "in the word: ""txt$"""
  PRINT
  PRINT "Uppercase word: "UCASE$(txt$)
END FN

LOCAL FN getInput
  PRINT
  PRINT "Press <return> to quit"
  INPUT "Enter a word: "; myText$
  LONG IF LEN(myText$) > 0
    FN doWord(myText$)
  END IF
END FN = LEN(myText$)

'--------- Main program ---------
FN initializeWindow

DO
  num = FN getInput
UNTIL num = 0
```

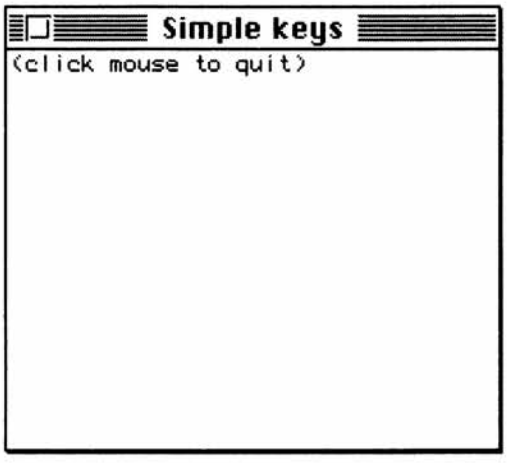
This gets the ASCII code of the first character and prints the character at the same time.

Curly-quotes.

If you press the return key, *myText$* has no character in it so it has a length of zero. When passed back as zero, the main loop exits.

Exercise 13.4

Type in the program below, run it, and fill in what is missing in the window:




```
'--------- Constants --------
_backSpace = 8
_cReturn   = 13
_space     = 32
_zLC       = 122


'--------- Functions --------
LOCAL FN buildWindow
  WINDOW 1,"Simple keys",(0,0)-(210,170)
  TEXT   _monaco, 9,,_srcCopy

  PRINT  "(click mouse to quit)"
  PRINT  %(0,32);     'set cursor
END FN


LOCAL FN checkOtherKeys (key$, h,v)

 LONG IF ASC(key$) = _backSpace
    PRINT %(h-6,v);" ";
    PRINT %(h-6,v);
  END IF

  LONG IF ASC(key$) = _cReturn
    PRINT %(0,v+11);
  END IF

END FN


LOCAL FN showTextCursor(h,v)
  PRINT %(h,v);"_";
  DELAY 20
  PRINT %(h,v);" "
  PRINT %(h,v);
END FN
```

The _srcCopy mode sets text to print opaque. This is required because text printing defaults to _srcOR, which is transparent. Opaque characters are necessary so the ones underneath them are erased.

If the backspace key is pressed, this backs up the cursor (where the next character will be printed) 6 pixels, the width of a character, and prints a space over it. It won't back up to the previous line because I didn't have enough room to add that routine.

This line moves the cursor down to the beginning of the next line if the *return* key is pressed. A nine point character is about 11 pixels high.

This function flashes a little underline so you know where the cursor is. It delays 20 milliseconds so you can see the underline, erases it, then backs up the cursor to where the next character will print.

continued next page...

continued from the previous page...

```
LOCAL FN printKeyPressed
  key$ = INKEY$
  h    = WINDOW(_penH)
  v    = WINDOW(_penV)

  LONG IF LEN(key$)
    LONG IF ASC(key$) >=_space AND ASC(key$) <=_LCz
      PRINT key$;
   'XELSE
      FN checkOtherKeys (key$, h,v)
    END IF
  XELSE
    FN showTextCursor(h,v)
  END IF

END FN

'--------- Main program ---------
FN buildWindow

DO
  FN printKeyPressed
UNTIL FN BUTTON
```

`INKEY$` checks if a key has been pressed. If one has, it puts the character into the *key$* variable.

These window functions get the position where the next character will be printed. These positions are used by both *checkOtherKeys* and *showTextCursor*.

If the ASCII code of the key pressed is in the range of printable characters, it prints the character. Look at the table at the beginning of this lesson to see character codes.

Programming problems

Program problem 13.1

Write a program that uses `INKEY$` to get a character from the keyboard.

When a key is pressed, display the character in the center of the window at points sizes from 4 to 48. The window output should look like a growing character. Quit the program when the mouse is clicked.

Program problem 13.2

Write a program that converts any string entered at the keyboard, to all lowercase letters.

Lesson 14

Reading data in memory

This lesson shows you how to store lists of numeric and text data in memory and how to read that data into variables.

DATA	This statement, followed by a list of numbers or character strings, stores the data in memory for later retrieval.
READ	This statement reads numeric or text data from the current position in DATA statements, into a specified variable or list of variables. After data is read, the pointer is moved to the next data item, so the next READ statement will read the next item.
RESTORE	Positions the data pointer to the first DATA item. This is required if you want to read the data again from the beginning.
RESTORE *n*	Positions the data pointer to the *nth* DATA item. This allows you to read data in any order. That is, you could read data from DATA item 10, item 3, item 22, and so on, in any order, by just repositioning the data pointer.

Exercise 14.1

Type in the program below, run it, and fill in what is missing in the window:

```
'--------- Data statements  ----
DATA "Sam Snead"
DATA "1234 E. Main Street"
DATA "St. Marcus, LA 32323
DATA "923-888-8888"
DATA " " .
DATA "Mike Smits"
DATA "9876 S. Oakland Ave.
DATA "San Francisco, CA 92312"
DATA "800-555-1212"
DATA " "
DATA "end"

'--------- Functions --------
LOCAL FN initializeWindow
  WINDOW 1,"DATA",(0,0)-(170,170)
  TEXT _monaco, 9          'window font
END FN

LOCAL FN printData
  DO
    READ  item$
    PRINT item$
  UNTIL item$ = "end"
END FN

'--------- Main program ---------
FN initializeWindow        'set up window
FN printData

DO
UNTIL LEN(INKEY$) OR FN BUTTON
```

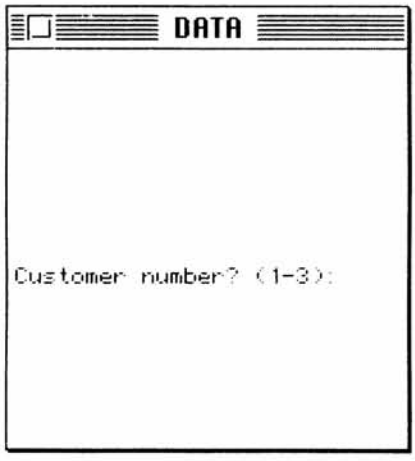
Data is stored at the top of a program. All the program's functions can see data in DATA statements.

The first time this line is executed, it reads the first data item, the second time through the loop it reads the second data item, and so on.

This loop quits when the string data "end" is encountered. This is a good way to know when you are finished reading all the items.

Exercise 14.2

Type in the program below, run it, and fill in what is missing in the window:




```
'--------- Data statements  ----
DATA "Sam Snead"
DATA "1234 E. Main Street"
DATA "St. Marcus, LA 32323
DATA "923-888-8888"
DATA "end"
DATA "Mike Smits"
DATA "9876 S. Oakland Ave.
DATA "San Francisco, CA 92312"
DATA "800-555-1212"
DATA "end"
DATA "Mary Muffit"
DATA "123 S. Tuffit"
DATA "Fairy, Gooseland 92312"
DATA "212-212-1212"
DATA "end"

'--------- Functions --------
LOCAL FN initializeWindow
  WINDOW 1,"DATA",(0,0)-(170,170)
  TEXT _monaco, 9
END FN

LOCAL FN printCustomer (num)
  item$ = ""
  RESTORE (num-1) * 5
  CLS: PRINT
  PRINT"Customer number:"num
  WHILE item$ <> "end"
    PRINT item$
    READ  item$
  WEND
END FN

LOCAL FN getCustomer
  PRINT%(3,100);: CLS LINE
  INPUT "Customer number? (1-3):";num
  LONG IF num > 0 AND num < 4
    FN printCustomer(num)
  XELSE
    PRINT "No such customer!" : BEEP
  END IF
END FN = num

'--------- Main program ---------
FN initializeWindow
DO
  num = FN getCustomer
UNTIL num = 0
```

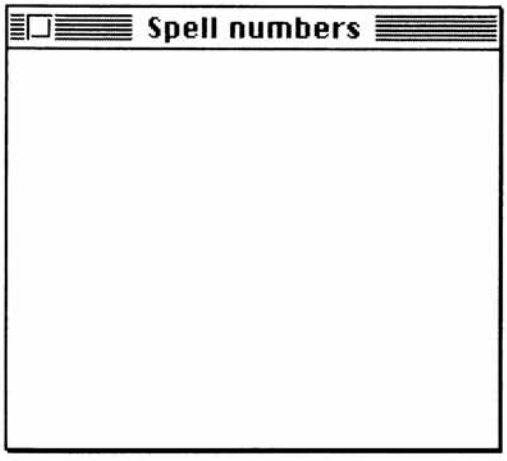
This is similar to the last exercise except that each data record used terminates with an "end' field.

This line calculates the start of each record (they all have five lines of data) and puts the data pointer at that point.

This clears text from the cursor position to the end of the line.

Exercise 14.3

Type in the program below, run it, and fill in what is missing in the window:




```
'--------- Data statements  ----
DATA "One",  "two", "three"
DATA "four",  "five", "six", "seven"
DATA "eight", "nine", "ten"

'--------- Sub-programs --------
LOCAL FN initializeWindow
  WINDOW 1,"Spell numbers",(0,0)-(210,170)
  TEXT _geneva, 9          'window font
END FN

LOCAL FN getSpell$(num)
  spell$ = "(no data)"
  LONG IF num > 0 AND num <= 10
    RESTORE num - 1        'point to cust
    READ spell$
  END IF
END FN = spell$

LOCAL FN spellNumber
  PRINT
  PRINT "(press zero to exit)"
  INPUT "Input a number (1-10): ";num
  PRINT "The number"num"is spelled: ";
  PRINT FN getSpell$(num)
END FN = num

'--------- Main program ---------
FN initializeWindow
DO
  num = FN spellNumber
UNTIL num = 0
```

One has to be subtracted because the first data item is zero, the second one is one, and so on.

Programming problems

Program problem 14.1

Write a program that stores the names of all the months in DATA statements. Print a column containing the numbers 1 through 12 corresponding to the month printed to the right of it.

Lesson 15
Files

This lesson shows you how to store data to a disk file and how to read that data back into variables in your program.

`OPEN` Opens a file so you can read or write to it. The syntax for this statement, followed by an explanation of each parameter:

```
OPEN "type", fileNum%, "filename",, vol%
```

type Types: "O" means open file for *Output* (write only). "I" means open file for *Input* (read only). "R" means open the file for *read and write*.

fileNum% Each open file is assigned a number. `READ`, `WRITE` and `CLOSE` need to know which file is being referred to, since several files can be open at the same time.

"filename" The file's name. It can be a quoted string or a string variable.

vol% This is the file's volume number (the number of the folder the file is in). If you don't use a volume, *FutureBASIC* expects the file to be in the same folder as the current program.

`READ# n` This reads data from a disk file *n*, previously written with `WRITE`, into a list of variables. When reading string variables you need to specifiy the number of characters.

```
READ#1, myString$;33
```

The line above reads 33 characters into the string variable *myString$*.

`WRITE# n` This statement, followed by a list of numeric or string variables, stores data to disk file *n*. When writing string variables you need to specify the number of characters.

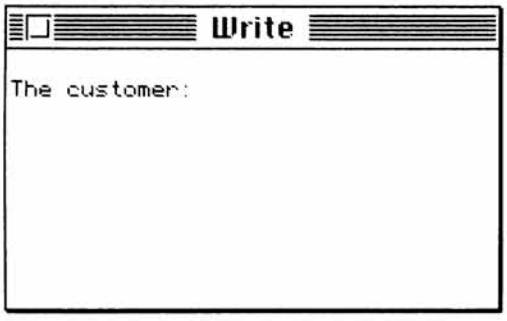
`RECORD# n` Similar to `RESTORE` in the last lesson. The `RECORD` statement positions the file pointer within file *n*.

`FILES$` This special function is used to bring up the standard open file and save file dialogs that people expect when they choose the *Open...* or *Save...* items from the File menu.

`CLOSE n` Closes file number *n*.

Exercise 15.1

Type in the program below, run it, and fill in what is missing in the window:



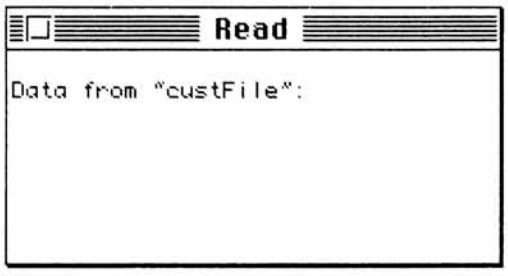

```
'--------- Functions --------
LOCAL FN initializeWindow
  WINDOW 1,"Write",(0,0)-(220,110)
  TEXT _monaco, 9
END FN

LOCAL FN writeData       'to disk
  cName$   = "John Jones"
  address$ = "123 E. 42nd Street"
  city$    = "New York"
  state$   = "NY"
  ZIP&     = 01234

  OPEN "O",1,"custFile"
    WRITE #1, cName$;10, address$;20
    WRITE #1, city$;10, state$;2, ZIP&
  CLOSE #1                'close file

  PRINT
  PRINT "The customer:"  'print data
  PRINT
  PRINT "   "cName$
  PRINT "   "address$
  PRINT "   "city$", "state$" "ZIP&
  PRINT
  PRINT "Was saved to the file: CustFile";

END FN

'--------- Main program ---------
FN initializeWindow
FN writeData

DO
UNTIL FN BUTTON OR LEN(INKEY$)
```

The file is opened for *output* only. Note that data is *written* to disk in exactly the same format as the program on the opposite page *reads* it.

Numbers following string variables specify how many characters to write to disk. Numeric variables, like ZIP&, don't need numbers after them.

Exercise 15.2

Type in the program below, run it, and fill in what is missing in the window:



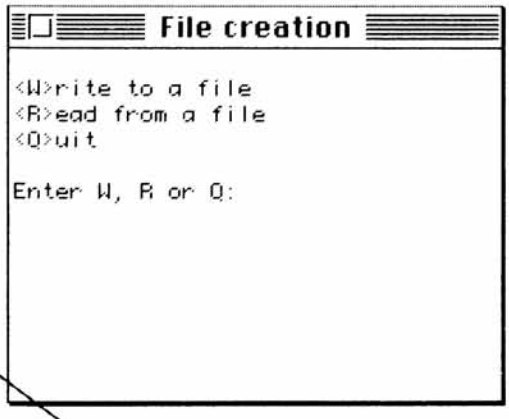

```
'--------- Functions --------
LOCAL FN initializeWindow
  WINDOW 1,"Read",(0,0)-(220,90)
  TEXT _monaco, 9
END FN

LOCAL FN readData        'from disk

  OPEN "I", 1, "custFile"
    READ #1, cName$;10, address$;20
    READ #1, city$;10, state$;2, ZIP&
  CLOSE #1

  PRINT
  PRINT "Data from "custFile":"
  PRINT
  PRINT cName$
  PRINT address$
  PRINT city$", "state$" "ZIP&
END FN

'--------- Main program ---------
FN initializeWindow
FN readData

DO
UNTIL FN BUTTON OR LEN(INKEY$)
```

The file is opened for *input* only.

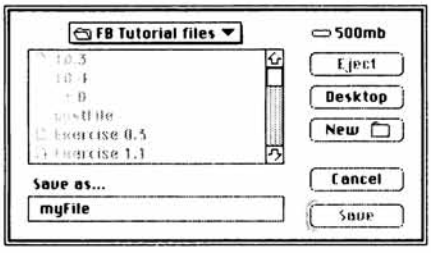
Exercise 15.3

Type in the program below, run it, and answer the questions at the bottom of the page:

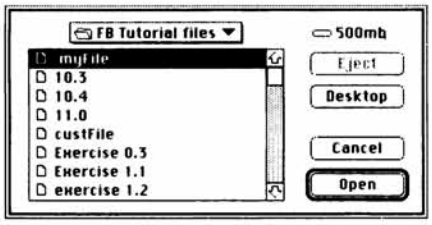
```
'------------- Functions ------------
LOCAL FN initializeWindow
  WINDOW 1,"File creation",(0,0)-(220,150)
  TEXT _monaco, 9
END FN

LOCAL FN saveFileAs$
  filename$ = FILES$(_fSave,"Save as...","",vol%)
  LONG IF LEN(filename$)=0  'cancel?
    filename$ = "cancelled"
  XELSE
    OPEN "O",1,filename$,,vol%
  END IF
END FN = filename$

LOCAL FN openFile$
  filename$ = FILES$(_fOpen,,,vol%)
  LONG IF LEN(filename$)=0  'cancel?
    filename$ = "cancelled"
  XELSE
    OPEN "I",1,filename$,,vol%
  END IF
END FN = filename$

LOCAL FN readDataFile
  filename$ = FN openFile$
  LONG IF filename$ <> "cancelled"
    CLS: PRINT
    PRINT "File name: "filename$
    PRINT "Contains the following data:"
    FOR count = 1 TO 5
      READ #1, fileData$;20, counter%
      DEF TRUNCATE(fileData$)
      PRINT fileData$" "counter%
    NEXT
    CLOSE #1
  END IF
END FN
```



This function uses FILES$ to call up the standard save file dialog box like the one shown below:



This function uses FILES$ to call up the standard open file dialog box like the one shown below:



This function reads data from a file saved with *writeDataFile*. The DEF TRUNCATE statement removes spaces from the end of strings read from disk.

continued next page...

```
LOCAL FN writeDataFile
  filename$ = FN saveFileAs$
  LONG IF filename$ < > "cancelled"
    FOR count = 1 TO 5
      WRITE #1, filename$;20, count
    NEXT
    CLS
    PRINT "Finished writing data to: "filename$
    CLOSE #1
  END IF
END FN

LOCAL FN getInput$
  CLS: PRINT
  PRINT "<W>rite to a file"
  PRINT "<R>ead from a file"
  PRINT "<Q>uit"
  PRINT
  PRINT "Enter W, R or Q:"
  DO
    task$ = UCASE$(INKEY$)
  UNTIL LEN(task$) > 0
END FN  = task$

LOCAL FN doTask(task$)
  quit = _false
  LONG IF task$ = "R"
    FN readDataFile
  END IF
  LONG IF task$ = "W"
    FN writeDataFile
  END IF
  LONG IF task$ = "Q"
    quit = _true
  END IF
  LONG IF task$ = "W" OR task$ ="R"
    PRINT
    PRINT "Click mouse to continue..."
    DO
    UNTIL FN BUTTON
  END IF
END FN = quit

'------------- Main program -------------
FN initializeWindow

DO
  task$ = FN getInput$
  quit  = FN doTask(task$)
UNTIL quit = _true
```

This is obviously an archaic way of getting instructions from a user. The menu section of this book will explain a more Mac-like way later.

This loop waits for a key to be pressed, then passes back the uppercase character of the task to be completed to the main program.

This function does the task specified by the key pressed in the *getInput* function.

Programming problems

Program problem 15.1

Write a program that opens a file and writes 100 random numbers to it.

Use `FILES$` so the user can specify the file name and where the file will go.

Be sure to close the file and print a message that says the program is finished.

Program problem 15.2

Write a program that opens a file created by the *Program problem 15.1*.

Use `FILES$` so the user can specify what file they want to open.

Print all the numbers in the file and wait for mouse click to quit. Format the numbers with `USING`.

To learn more about file handling with FutureBASIC, read the File handling section of the Handbook.

Lesson 16

Array variables

Arrays are a special type of multiple variable. Array variables contain multiple variables of some type: string, integer, floating, etc. Each variable in the array is numbered in consecutive order. Array variables are defined with a `DIM` statement.

The following program line creates a character string array with six elements, *myNames$(0)* through *myNames$(5)*:

```
DIM myName$(6)      'create 6-elements
```

The reason array variables are so useful is that that can be accessed by a number, somewhat like a cell in a spreadsheet can be accessed by a number.

Imagine trying to print 1,000 names in 1000 different variables! You would have a thousand print lines like this:

```
PRINT name1$
PRINT name2$
PRINT name3$
```

and type another 997 lines...

A much easier way is to use an array and print all the names in the array in a simple loop like the example below:

```
FOR count = 1 to 1000    'print all names
   PRINT myName$(count) 'from 1 to 1000
NEXT
```

Arrays can also be used for sorting large lists of data. Names or numbers in an array can be moved around until they are in order. For example:

Array *before* sorting:

names$(0)	= Mike
names$(1)	= Alan
names$(2)	= Frank
names$(3)	= Tom
names$(4)	= Peter

Array *after* sorting:

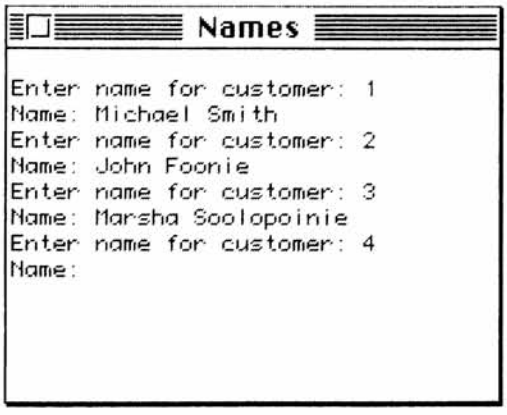
names$(0)	= Alan
names$(1)	= Frank
names$(2)	= Mike
names$(3)	= Peter
names$(4)	= Tom

The capability of accessing, sorting and printing data numerically makes array variables indispensable.

Exercise 16.1

This program shows you how to access names in an array by number. Type in the program below, run it, and fill in what is missing in the window:

```
'----------- Global variables ------
DIM myName$(5)
END GLOBALS

'------------ Functions -------------
LOCAL FN initializeWindow
  WINDOW 1,"Names",(0,0)-(220,150)
  TEXT _monaco, 9
END FN

LOCAL FN getNames
  PRINT
  myName$(0) = "goodbye!"
  FOR count = 1 TO 5
    PRINT "Enter name for customer:"count
    INPUT "Name: ";myName$(count)
  NEXT
END FN

LOCAL FN showName(num)
  LONG IF num >=0 AND num <=5   'in range?
    PRINT "Customer"num"is "myName$(num)
    PRINT
  END IF
END FN

'--------- Main program ---------
FN initializeWindow
FN getNames

DO
  INPUT "Enter customer number (1-5):";cust
  FN showName(cust)
UNTIL cust = 0
```



This is the "Global variable" section of the program. Global variables should be defined at the top of the program. Put END GLOBALS on the line following the last global variable. Global variables can be seen and modified by all functions in a program. Too many globals make a program harder to debug.

This program dimensions a six element array for holding six strings. Actually, only elements one through five are used. Element zero is not used.

Exercise 16.2

This program uses arrays to sort data. Type in the program below, run it, and fill in what is missing in the window (make your own sketch of the window since it won't fit here):

```
'--------- Global variables --------
DIM array(20)
END GLOBALS

'--------- Functions ---------------
LOCAL FN initializeWindow
  WINDOW 1,"Sorting",(0,0)-(270,250),_docNoGrow
  TEXT _monaco, 9
END FN

LOCAL FN printArray (horzPos, message$)
  PRINT %(horzPos,20);message$
  FOR count = 0 TO 20
    PRINT %(horzPos,(count+3)*10);
    PRINT "array(";
    PRINT USING "##";count;")= ";
    PRINT USING "##,###";array(count)
  NEXT
END FN

LOCAL FN fillArrayWithRandomNums
  FOR count = 0 TO 20
    array(count) = RND(20000)
  NEXT
  FN printArray(20,"-- BEFORE SORT --")
END FN

LOCAL FN sortArray
  FOR current = 0 TO 20
    FOR look = current TO 20
      LONG IF array(current) >= array(look)
        SWAP array(current), array(look)
      END IF
    NEXT
  NEXT
  FN printArray(145,"-- AFTER SORT ---")
END FN

'---------- Main program ----------
FN initializeWindow
FN fillArrayWithRandomNums
FN sortArray

DO
UNTIL FN BUTTON
```

This program dimensions a 21 element array for sorting 21 integers.

Provide the horizontal position and this function prints the entire array in a column. This function is used to print the array both before and after it is sorted, so you can see the result of the sort.

Change ">=" to "<=" and the array will be sorted high to low, instead of low to high.

This statement swaps the contents of the two variables. This results in the higher numbers "bubbling" up to the top of the array.

Exercise 16.3

This program is essentially the same as the previous exercise except that a string array is used instead of numbers.

Type in the program below, run it, and make a sketch of the window:

```
'--------- Global variables ----------
DIM array$(20)
END GLOBALS

'--------- Functions ----------------
LOCAL FN initializeWindow
  WINDOW 1,"Sort$",(0,0)-(270,250),_docNoGrow
  TEXT _monaco, 9
END FN

LOCAL FN printArray (horzPos, message$)
  PRINT %(horzPos,20);message$
  FOR count = 0 TO 20
    PRINT %(horzPos,(count+3)*10);
    PRINT "array$(";
    PRINT USING "##";count;")= ";
    PRINT array$(count)
  NEXT
END FN

LOCAL FN fillArrayWithRandomChars
  FOR count = 0 TO 20
    char$ = ""
    FOR letter = 1 TO 5
      char$ = char$ + CHR$(RND(26) + 64)
    NEXT
    array$(count) = char$
  NEXT
  FN printArray(20,"-- BEFORE SORT --")
END FN

LOCAL FN sortArray
  FOR current = 0 TO 20
    FOR look = current TO 20
      LONG IF array$(current) >= array$(look)
        SWAP array$(current), array$(look)
      END IF
    NEXT
  NEXT
  FN printArray(145,"-- AFTER SORT ---")
END FN

'---------- Main program -------------
FN initializeWindow
FN fillArrayWithRandomChars
FN sortArray

DO
UNTIL FN BUTTON
```

This program dimensions a 21 element array for sorting 21 strings.

Change ">=" to "<=" to sort from Z to A instead of A to Z.

Exercise 16.4

Look at the program below, run it, and fill in what is missing in the window.

```
'-------- Global variables ---------
DIM result&(12)          'hold dice 2-12
END GLOBALS

'-------- Functions ----------------
LOCAL FN initializeWindow
  WINDOW 1,"Dice Simulation",(0,0)-(240,210)
  TEXT _monaco, 9
END FN

LOCAL FN printResult(count&)
  CLS: PRINT
  PRINT "Total dice rolls so far: "count&"
  PRINT
  PRINT "Dice roll      Total rolls     Odds%"
  FOR x = 2 TO 12      'print rolls
    PRINT "    ";      'space over
    USING "##";x,
    PRINT USING "##,###";result& (x),
    PRINT USING "##%";result&(x)/count&*100
  NEXT
  PRINT
  PRINT "Click mouse to quit...";
END FN

LOCAL FN getDiceRoll (count&)
  die1 = RND(6)
  die2 = RND(6)
  roll = die1 + die2
  result&(roll) = result&(roll) + 1
  LONG IF count& MOD 1000 = 0
    FN printResult(count&)
  END IF
END FN

'-------- Main program ------------
FN initializeWindow

DO
  count& = count& + 1
  FN getDiceRoll (count&)
UNTIL LEN(INKEY$) OR FN BUTTON
```

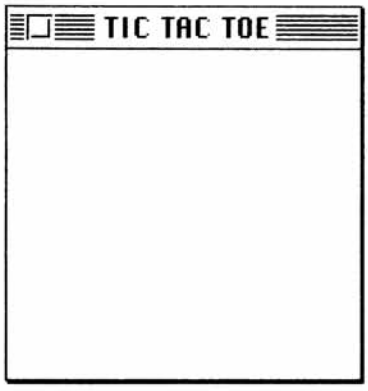
This function displays the distribution of dice rolls every 1,000 rolls. The more times the dice are rolled, the more accurate the odds become.

These lines generate two random numbers, each representing a six-sided die with numbers one through six on each side. The total of the two dice is put in the *roll* variable.

If a twelve is rolled, then array element 12 is incremented by one. This keeps track of how many times 12 was rolled. The same goes with the other possible dice rolls of 2 through 11.

Exercise 16.5

Look at the program below, run it, and fill in what is missing in the window.




```
'---------- Constants -------------
_xPiece = 88              '88 = ASC("X")
_oPiece = 79              '79 = ASC("O")


'---------- Global variables ------

'   1  |  2  |  3
'  ---------------
'   4  |  5  |  6
'  ---------------
'   7  |  8  |  9

DIM board(9)
END GLOBALS


'----------- Functions ------------
LOCAL FN initializeWindow
  WINDOW 1,"TIC TAC TOE",(0,0)-
(150,140),_docNoGrow
  TEXT _geneva, 24
END FN

LOCAL FN drawLines
  PLOT 10,44 TO 140,44
  PLOT 10,92 TO 140,92
  PLOT 50,10 TO 50,130
  PLOT 90,10 TO 90,130
END FN

LOCAL FN drawPiece (sq)
  down   = (sq-1) / 3
  across = ((sq+2) MOD 3)

  LONG IF board(sq)>0
    PRINT %((across*40)+23, (down*43)+35);
    PRINT CHR$(board(sq));
  END IF
END FN
```

The tic-tac-toe board positions are held in a nine element array. This grid shows which element holds the piece for which square. For example square one is stored in element 1. Each element holds the piece for that square (zero if no piece on the square).

This function calculates where a piece goes on the tic-tac-toe board. MOD returns the remainder of a divide to determine which column the piece is on. For example, 8 divided by 3 has a remainder of 2, meaning square 8 is the second square over.

The ASCII value of the character is stored in the array as _xPiece or _oPiece, representing the letters the "X" or "O".

```
LOCAL FN setUpBoard
  'put a few pieces on the board
  board(1) =_xPiece: board(2)=_oPiece
  board(3) =_xPiece: board(4)=_oPiece
  board(6) =_xPiece: board(9)=_oPiece
END FN

LOCAL FN drawBoard
  FN drawLines        'grid lines
  FN setUpBoard       'put pieces in array

  FOR square = 1 TO 9'put piece on board
    FN drawPiece (square)
  NEXT

END FN

'-------- Main program ------------
FN initializeWindow
FN drawBoard

DO
UNTIL FN BUTTON OR LEN(INKEY$)
```

This routine sets up the board by putting pieces into the array.

Programming problems

Program problem 16.1

Write a program that dimensions a 12 element string array that holds the name of each of the 12 months, starting with January. Write a function that prints all the months from the array in order.

Program problem 16.2

Modify exercise 16.2 to create a 2000 element array instead of just 20. Load 2000 random numbers into the array and sort the array. Print only the last 20 elements of the array. Use `TIME$` to time the sort.

Lesson 17

Selecting one case out of many

When a program needs to compare a value against more than two cases, the LONG IF statement becomes tedious and hard to read.

SELECT/CASE provides a way to compare a value against many cases.

```
SELECT myValue
  CASE 10
    PRINT "myValue = 10"
  CASE 200
    PRINT "myValue = 200"
  CASE 3000
    PRINT "myValue = 3000"
  CASE ELSE
    PRINT "myValue ="myValue
END SELECT
```

In the example above, "SELECT myValue" tells the computer to compare *myValue* to the values after each CASE statement below it. If *myValue* equals 200, *myValue = 200* would be printed and the program would continue on the next line. If *myValue* equalled 2220, it would fall through to the CASE ELSE line and print that.

There is a second form of SELECT that allows you to select among more numerous conditions even if they do not involve the same variable. This form also allows condition tests other than equality.

```
SELECT
  CASE myValue = 100, myValue = 200
    PRINT "myValue = 100 or 200"
  CASE yourValue = 99 + myValue * 100
    PRINT "YourValue works for me!"
  CASE myValue = 950 - yourValue
    PRINT "myValue worked OK!"
  CASE ELSE
    PRINT "nothing above works for me!"
END SELECT
```

One SELECT/CASE statement is far easier to read than many LONG/IF or IF statements.

Exercise 17.1

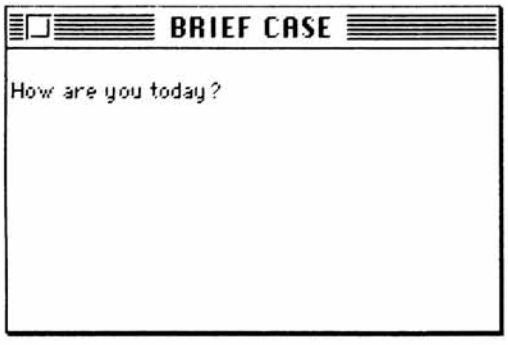
Look at the program below, run it, and fill in what is missing in the window.

```
'--------- Constants -----------
_printHello   = 1       'Define a list
_beepOnce     = 2       'of commands for
_beepTwice    = 3       'the program
_printGoodbye = 4       'to do

'--------- Functions --------
LOCAL FN initializeWindow
  WINDOW 1,"Do command",(0,0)-(150,120)
  TEXT _geneva, 9,,_srcCopy
  PRINT
  PRINT "<1> Print Hello"
  PRINT "<2> Beep once"
  PRINT "<3> Beep twice"
  PRINT "<4> Say goodbye and quit"
END FN

LOCAL FN doCommand
  quit = _false
  what = VAL(INKEY$)
  SELECT what
    CASE _printHello
      PRINT %(5,80) "Hello!";
    CASE _beepOnce
      BEEP
    CASE _beepTwice
      BEEP:BEEP
    CASE _printGoodbye
      PRINT %(5,80) "Goodbye!"
      quit = _true
      DELAY 1000
  END SELECT
END FN = quit

'--------- Main program ---------
FN initializeWindow

DO
  quit = FN doCommand
UNTIL FN BUTTON OR quit = _true
```

The variable *what* is compared to the value after each CASE statement.

Exercise 17.2

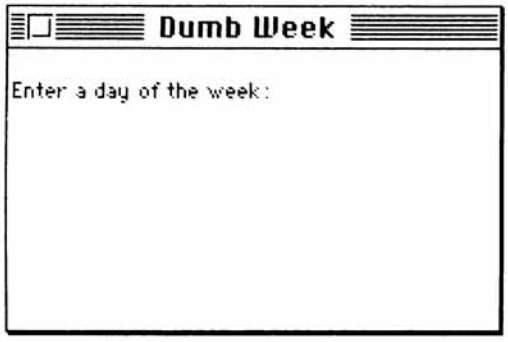
Look at the program below, run it, and fill in what is missing in the window.




```
'--------- Functions --------

LOCAL FN initializeWindow
  WINDOW 1,"BRIEF CASE",(0,0)-(210,120)
  TEXT _geneva, 9
END FN

LOCAL FN getResponse$ (what$)

  SELECT UCASE$(what$)
    CASE "GREAT", "AWESOME", "WONDERFUL"
      respond$ = "I'm excited to hear that!"

    CASE "OK", "GOOD", "ALLRIGHT", "WELL"
      respond$ = "That's nice."

    CASE "AWFUL", "BAD", "SAD", "MAD"
      respond$ = "I'm sorry to hear that!"

    CASE ELSE
      respond$ = "Hmmm...."
  END SELECT

END FN = respond$

LOCAL FN askQuestion
  PRINT
  INPUT "How are you today? ";ask$
  PRINT FN getResponse$ (ask$)
END FN

'--------- Main program ---------
FN initializeWindow
FN askQuestion

DO
UNTIL FN BUTTON
```

Exercise 17.3

Look at the program below, run it, and fill in what is missing in the window.




```
'--------- Functions --------
LOCAL FN initializeWindow
  WINDOW 1,"Dumb Week",(0,0)-(210,120)
  TEXT _geneva, 9
END FN

LOCAL FN weekResponse$ (day$)

  SELECT UCASE$(day$)
    CASE "MONDAY"
      respond$ = "Mondo day"
    CASE "TUESDAY"
      respond$ = "day before tuesmorrow"
    CASE "WEDNESDAY"
      respond$ = "Wedding nest day"
    CASE "THURSDAY"
      respond$ = "Thirst day"
    CASE "FRIDAY"
      respond$ = "Fried day"
    CASE "SATURDAY"
      respond$ = "Sad bird day"
    CASE "SUNDAY"
      respond$ = "Son day"
    CASE ELSE
      respond$ = "That's not a day!"
  END SELECT

END FN = respond$

LOCAL FN askQuestion
  CLS
  PRINT

  INPUT "Enter a day of the week: "; ask$

  PRINT FN weekResponse$ (ask$)
  PRINT
  PRINT "Click mouse to quit or wait"
  PRINT "for another question..."
END FN
```

continued on the next page...

continued from the previous page...

```
LOCAL FN checkMouseButton
  quit = _false
  time = VAL(RIGHT$(TIME$,2))

  DO
    seconds = ABS(time-VAL(RIGHT$(TIME$,2)))
    IF FN BUTTON THEN quit = _true
  UNTIL quit = _true OR seconds > 3
END FN = quit

'--------- Main program ---------
FN initializeWindow

DO
  FN askQuestion
  quit = FN checkMouseButton
UNTIL FN BUTTON
```

This function waits three seconds for you to quit by clicking the mouse. If you don't click the mouse in time, it asks you another question.

Programming problems

Program problem 17.1

Modify Exercise 17.1 by adding a "Beep three times" item and a "Print Good Day" item.

Lesson 18

ResEdit and resources

Many people are under the mistaken impression that resources are some kind of secret code known only by the high-priests of programming. This simply isn't true.

Resources, such as pictures, sounds, text, and others, can be easily incorporated in programs using the RESOURCES statement.

There are many types of resources you can use in your programs. You can include pictures, sounds, cursors, among others that can:

- Create pictures in a paint or draw program, paste them into your program's resource file, then easily display them in windows with *FutureBASIC's* PICTURE statement.
- Create sound effects using the *Sound control panel* under the Apple menu, copy and paste them into your resource file, then play them from your programs using *FutureBASIC's* SOUND statement.
- Create cursor shape resources (like the arrow and watch) that you can use in your programs. For example, a paint program would need a pencil cursor for drawing.

The RESOURCES statement tells *FutureBASIC* to add all the resources in that file to your program:

```
RESOURCES "resourceFile"
```

Resources are identified by a four-letter type, such as "PICT", and a numeric ID number, such as 128. You can also name a resource with a string, such as "myPicture". For example, if you have a picture resource of the type "PICT" with an id of 199, and also a name of "myPicture", you can access it two ways:

```
myPict& = FN GETRESOURCE("PICT", 199)
myPict& = FN GETNAMEDRESOURCE("PICT","myPicture")
```

The long-integer variable *myPict&* holds a handle to the picture. This handle would be used to draw the picture.

A special program for creating, editing, and modify resources and resource files is included with *FutureBASIC* called *ResEdit.* The exercises in this lesson will familiarize you with *ResEdit* and show you how simple it is to incorporate resources in your programs.

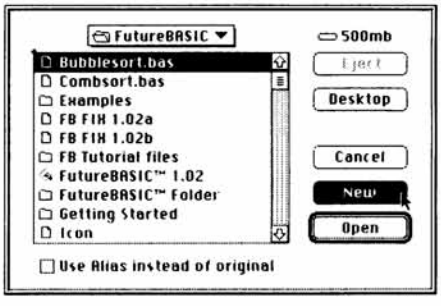
Exercise 18.1

Before we create a picture, we will create a resource file in *ResEdit* called *myFile.rsrc* that will be used in a subsequent exercise. Follow these steps:

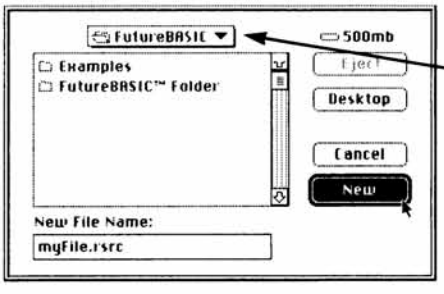
1. Find and open *ResEdit.*

It is on the *ResEdit and MacsBug* disk that came with *FutureBASIC*. If you haven't done so already, read the *Getting Started* manual for installing *ResEdit* on your hard disk.

2. Click away *ResEdit's* cute start-up window and you'll see a window like the one below. Click the *New...* button.

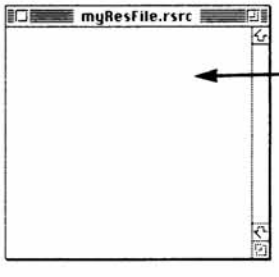


3. Give your resource file the name "myFile.rsrc" and click the "New" button.



When you create the new resource file, make sure it's in the same folder as your Exercise files. *FutureBASIC* requires that the resource file be in the same folder as your program.

4. You will now be in *ResEdit* and see a window like the one below. Proceed to the next exercise.

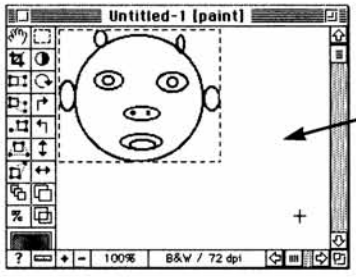


If your window looks slightly diferent from this one, go to the to *Views* menu in *ResEdit* and change it to "by Icon" view.

Exercise 18.2

In this exercise, you'll create an image and put it in the *myFile.rsrc* resource file as a "PICT" resource.

1. Load a graphics program and draw a face.



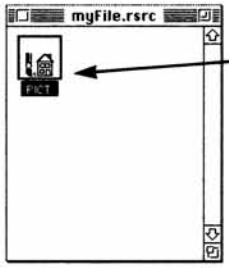
I used DeskPaint to draw my face, but you can use DeskDraw, MacDraw, MacPaint, Canvas, PhotoShop, SuperPaint or any other graphic program. (You will not be graded on artistic quality, only originality.)

If you are using a draw program, use the arrow tool instead of the marquee to select your drawing.

2. After you draw the face, select it with the marquee and copy it to the clipboard with *Copy* from the Edit menu.
3. Now go to *ResEdit* and make sure the *myFile.rsrc* file is open and its window is in front.

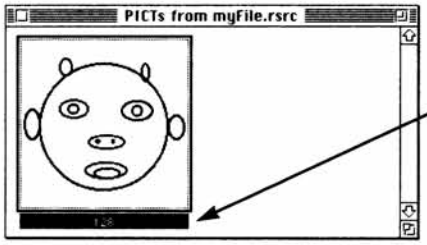
This is the same window shown at the end of *Exercise 18.1.*

4. Put the image into your resource file by choosing *Paste* from the Edit menu. The *myFile.rsrc* window will change to show that the file now contains a "PICT" resource.



The icon represents all the PICT resources in the file. If you paste another PICT resource into the file you will still see only one PICT icon. It works the same way with other resource types.

5. Double-click the PICT icon and you will see a window showing your picture as resource ID 128.



The picture you pasted in will have a default id of 128. You can change this by using *Get Resource Info...* from the Resources menu. For this exercise, leave it as id 128.

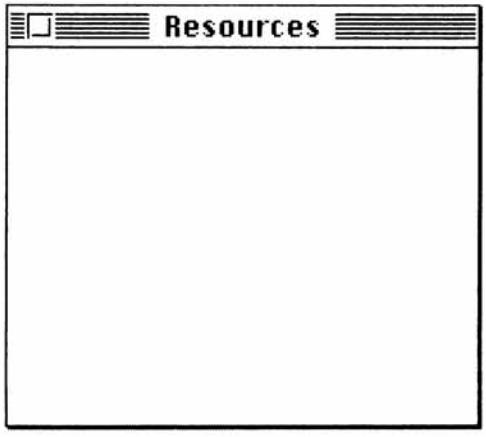
6. Choose *Save...* from the File menu. The resource file should now be in the same folder as your Exercise programs.
7. Be sure to close the *myFile.rsrc* window so you don't get errors from *FutureBASIC* when running your program.

Go to the next exercise.

Exercise 18.3

Look at the program below, run it, and fill in what is missing in the window.

```
RESOURCES "myFile.rsrc"

'--------- Functions --------
LOCAL FN initializeWindow
  WINDOW 1,"Resources",(0,0)-(210,160)
END.FN

LOCAL FN showPICT(pictNum)

  myPict& = FN GETRESOURCE(_"PICT", pictNum)

  LONG IF myPict& >< 0
    PICTURE (0,0), myPict&
  XELSE
    PRINT "ERROR!"          'check resource
    PRINT "PICT ERROR!"     'zero = error
    BEEP:BEEP
  END IF
END FN

'--------- Main program --------
FN initializeWindow
FN showPICT (128)

DO
UNTIL FN BUTTON
```



This line tells *FutureBASIC* to open the resource file called "myFile.rsrc" (which must be in the current folder) and use any resources it finds in it (PICT resource id 128 in this case).

This line gets the PICT resource with an id of *pictNum* (128 here). It puts the "handle" to this picture in the *myPict&* variable.

If *myPict&* does not equal zero, the picture will be displayed in the window. If it does equal zero, an error message will be displayed. If an error occurs, make sure the PICT resource has an id of 128 and that it has been pasted correctly into the resource file.

If you get errors from FutureBASIC saying the resource file could not be found, first make sure the resource file is in the same folder as your program, then check that the file is not still open in ResEdit. Finally, check that you do not have a virus detection program conflicting with FutureBASIC.

There are many more examples in the Sound resources, Graphic resources, Text resources, and Cursor resources chapters of the Handbook.

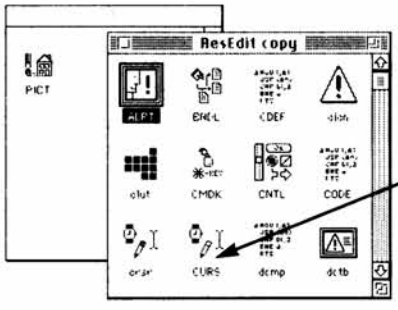
Exercise 18.4

This exercise will show you how to copy a resource from one resource file and paste it into another. Since resources often already exist in programs you've written or in *ResEdit*, this-procedure saves time.

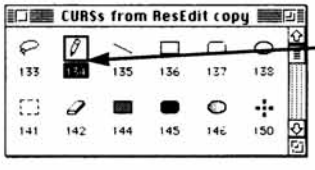
Any resource can be copied from one resource file and pasted into another. This exercise will show you how to copy a cursor resource of the type "CURS" from *ResEdit* and paste it into the *myFile.rsrc* resource file.

Follow the steps below:

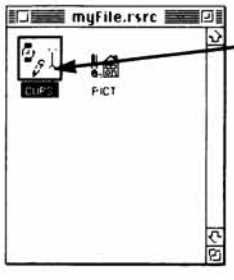
1. From *ResEdit*, open *myFile.rsrc* (the file you just created).
2. From *Finder*, make a duplicate of the *ResEdit* file.
3. From *ResEdit*, open *ResEdit copy* (the file just duplicated). You will see two windows that look something like this:



4. Since we want to copy a cursor resource from *ResEdit copy* and put it into *myFile.rsrc*, double-click the CURS icon to display a window showing all the cursor resources:



5. Select cursor id 134 (a pencil). Choose *Copy* from the Edit menu to put it on the clipboard.
6. Bring the *myFile.rsrc* window to the front. Choose *Paste* from the Edit menu. The window will look like this:



7. Save *myFile.rsrc* and quit *ResEdit*.

It's not a good idea to edit resources in files that are open. That's why we made a copy of the *ResEdit* file.

ResEdit, like most other Macintosh applications, has lots of resources. Many of them are commonly used resources. Later, when you're done with this exercise, take a look at resources in backups of some of your other applications: *Don't use originals.*

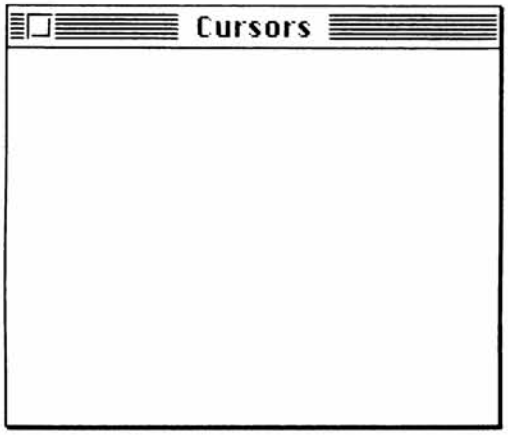
Here's the pencil cursor we want to copy over to our resource file. If you are writing a program that requires a special cursor (that's not already in *ResEdit),* use *ResEdit's* cursor editor to modify or create a new cursor resource. There are also editors for icons, menus, and many other resources.

ResEdit automatically adds a CURS resource icon for holding the pencil cursor.

If you leave *ResEdit* open, be sure to close the *myFile.rsrc* file. If it's still open, you 'll get an error from *FutureBASIC* when it tries to access it with RESOURCES.

Exercise 18.5

Look at the program below, run it, and fill in what is missing in the window.




```
RESOURCES "myFile.rsrc"

'--------- Constants -----------
_pencilCursor = 134

'--------- Functions --------
LOCAL FN initializeWindow
  WINDOW 1,"Cursors",(0,0)-(210,160)
END FN

LOCAL FN checkMouse
  LONG IF FN BUTTON
    CURSOR _pencilCursor
  XELSE
    CURSOR _arrowCursor
  END IF
END FN

'--------- Main program --------
FN initializeWindow

DO
  FN checkMouse
UNTIL LEN(INKEY$)     'press key to quit
```

Whenever the mouse button is down, this routine turns the cursor into a pencil. When you let up, it puts the cursor back to the standard arrow.

The CURSOR statement knows enough to look for a cursor resource so you don't have to use FN GETRESOURCE. There are several cursor resources always available to your programs. See CURSOR in the *Reference* manual and the *Cursor and Cursor resources* chapter of the *Handbook* for more information about using cursor resources.

Programming problems

Program problem 18.1

Read the *Sound resources* chapter of the *Handbook*. Enter the program under *Playing sounds*. Make sure that your program file and the *resfile.rsrc* resource file are in the same folder before running the program.

Program problem 18.2

Read the *Resources and ResEdit* chapter of the *Handbook*. Try doing some of the items under *Learning ResEdit* on the last page of the chapter.

Lesson 19

Events, buttons and menus

Events

All the following are events: menu item selection, button click, window size, close box click, scroll bar adjustment, zoom box click, window needing update, cursor moved over an edit field, and so on. Of course, there are many other kinds of events.

FutureBASIC programs have a central hub called an *event loop*. HANDLEEVENTS is the core of this central hub. The loop goes round and round waiting for an event. When one occurs, it calls the function you defined to handle that kind of event.

For example, a button click is a DIALOG type event. When a button you created is clicked, HANDLEEVENTS calls the function you defined to handle DIALOG events, with ON DIALOG FN doDialog below. The *doDialog* function determines what button was clicked and calls a function to handle it. Here's an outline of the way a program with event handling is organized:

```
'----- Functions -----
LOCAL FN doBreak
END FN

LOCAL FN doDialog
END FN

LOCAL FN filterKeyPress
END FN

LOCAL FN abortPrinting
END FN

LOCAL FN doMenus
END FN

LOCAL FN doMouse
END FN

LOCAL FN doTimer
END FN

'----- Main ------
ON BREAK    FN doBreak
ON DIALOG   FN doDialog
ON EDIT     FN filterKeyPress
ON LPRINT   FN abortPrinting
ON MENU     FN doMenus
ON MOUSE    FN doMouse
ON TIMER 5  FN doTimer

DO
   HANDLEEVENTS
UNTIL programEnds
```

When HANDLEEVENTSdetects an event, it calls the function defined by ON<*eventType*> to handle that kind of event.

A program has to tell HANDLEEVENTS which functions to call when certain types of events occur. Here are brief descriptions of which functions handle which event types:

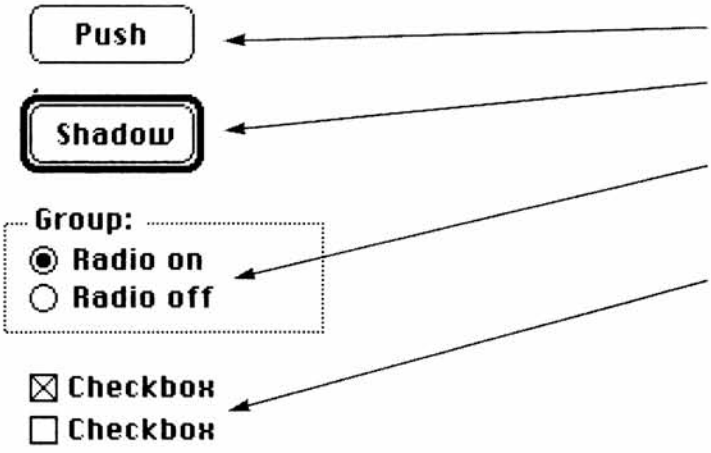
- cmd-period keys held down
- button click, window close, many others
- filters keys going to an edit field
- printing aborted with cmd-period
- menu selections
- mouse clicks
- timer events

This is the *Main event loop* where all events are detected and routed by HANDLEEVENTS.

Buttons

You will be shown how to create buttons and how to react to button clicks. A button event is generated whenever a button is clicked. Your program will determine which button was clicked and do what the button says it will do.

FutureBASIC provides these types of buttons:



Push button

Shadow button

Radio buttons are always in a group. If one radio button is marked, the other buttons in the group are unmarked.

Check boxes (not always in a group and not mutually exclusive).

A program draws a button in a window using the BUTTON statement. Each button has its own id. When the person using the program clicks a button, several things happen:

1. HANDLEEVENTS detects the button click.
2. It knows that button clicks are of the DIALOG type so it calls the function specified by the program to handle DIALOG events. In the example on the previous page, this was the responsibility of the *doDialog* function. If no function is designated to handle DIALOG events, the event is ignored.
3. The first thing the *doDialog function* does is get information about the event using these functions:

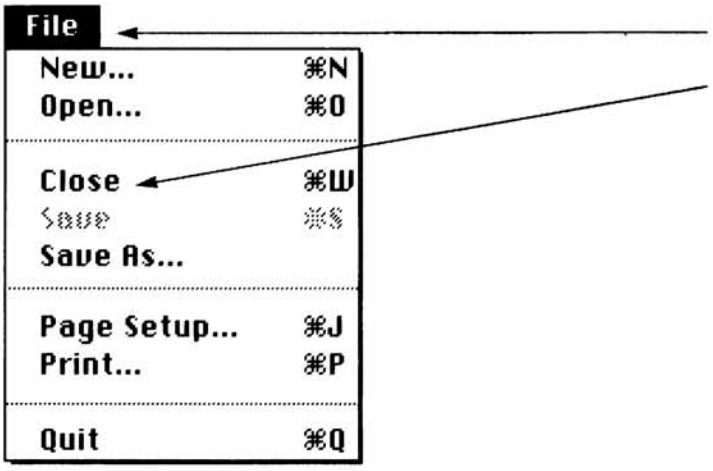
   ```
   evnt = DIALOG(0)
   id   = DIALOG(evnt)
   ```

   The variable *evnt* contains the value *_btnClick*, a constant telling us a button click event occurred. The *id* variable contains the button number.
4. The program then calls the function for handling that particular button. For example, if a button is called "Save", a function would be called to save the file.

The following exercises will show you how this is done for several types of buttons.

Menus

This lesson describes how to create menus and how to react to menu events. A menu event is generated when a menu item is chosen. Your program will determine which item of which menu was chosen and respond appropriately.



The File menu is usually menuID 1.

New... is itemID 1, Open... is itemID 2, Close is itemID 4, and so on.

Menu items can have key equivalents, can be gray so they can't be selected, can be checked to indicate that a mode is set, and more.

A program installs menus using the MENU statement. Each menu has its own id and one or more items. When the user selects a menu item, several things happen:

1. HANDLEEVENTS detects the menu selection.
2. It knows that menu selections are of the MENU type so it calls the function specified by the program to handle MENU events. In the example at the start of this lesson, this was the responsibility of the *doMenu* function. If no function is designated to handle MENU events, the event is ignored.
3. The first thing the *doMenu function* does is get information about the event using these functions:

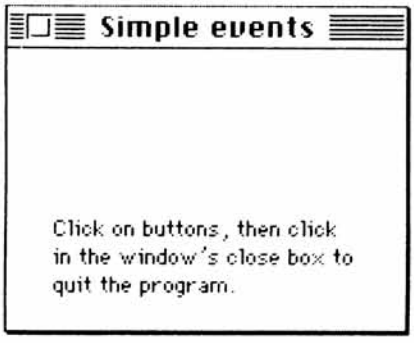
   ```
   menuID = MENU(_menuID)
   itemID = MENU(_itemID)
   ```

   The variable *menuID* contains the menu selected. The *itemID* variable contains the item in the menu.
4. The program then calls the function for handling that menu item. For example, if the "Quit" command was selected from the File menu, or ⌘-Q was pressed, the program would ask if the user wants to save changes before quitting.

The following exercises show how menus are created and how menu events are handled.

Exercise 19.1

Enter the program below, run it, and fill in what is missing in the window.




```
'------------ Global variables -------
DIM programEnds
END GLOBALS

'------------ Functions --------------
LOCAL FN initializeWindow
  WINDOW 1,"Events",(0,0)-(170,120),_docNoGrow
  TEXT _geneva, 9

  BUTTON 1,_activeBtn,"bt1",(20,10)-(150,30),_push
  BUTTON 2,_activeBtn,"bt2",(20,40)-(150,60),_push

  PRINT %(20,85) "Click buttons, then click"
  PRINT %(20,97) "the window's close box to"
  PRINT %(20,109)"quit the program.";
END FN

LOCAL FN doDialogEvnts
  evnt = DIALOG(0)
  id   = DIALOG(evnt)
  SELECT evnt
    CASE _btnClick
      LONG IF id = 1
        BEEP
      XELSE
        BEEP:BEEP
      END IF
    CASE _wndClose
      programEnds = _true
  END SELECT
END FN

'--------- Main program ---------
FN initializeWindow
ON DIALOG FN doDialogEvnts

DO
  HANDLEEVENTS
UNTIL programEnds
```

These two lines put push-buttons in the window with id's of 1 and 2.

When a button is clicked or a window is closed, HANDLEEVENTS calls this function as defined by ON DIALOG FN doDialogEvnts

evnt contains the *type* of event, such as a button click, window closed, etc.

id contains the id of the event, such as id of the button clicked, id of window closed, etc.

This tells HANDLEEVENTS what function to call when DIALOG type events occur. Both button clicks and window close events are DIALOG events. This line does not call the function directly. It only tells HANDLEEVENTS what function to call.

HANDLEEVENTS checks for events each time through the loop. When an event occurs, the doDialogEvnts function is called.

Program quits when *programEnds* equals _true or if you press the cmd-period keys.

Exercise 19.2

Enter the program below, examining the annotations as you do so. Then run the program to make sure it works.

```
'------------ Constants --------------
_fileMenu = 1          'file menu number
_fNewWnd  = 1          'new window item
_fQuit    = 3          'quit item

_beepMenu = 2          'beep menu number
_bBeep1   = 1          'beep once item
_bBeep2   = 2          'beep twice item

'------------ Global variables -------
DIM gWndNum
DIM gProgramEnds
END GLOBALS

'------------ Functions --------------
LOCAL FN openWindow (ttl$, id)
  LONG IF id < 65
    v = id * 6 + 36
    h = id * 6
    WINDOW id,ttl$,(h,v)-(170+h,60+v),_docNoGrow
    gWndNum = id + 1 'next wind to open

    BUTTON 1,_activeBtn,"beep3",(20,20)-(150,40),_push
  END IF
END FN

LOCAL FN putUpMenus
  'File menu ------
  MENU_fileMenu,0,_enable, "File"
  MENU_fileMenu,1,_enable, "/NNew Window..."
  MENU_fileMenu,2,_enable, "-"
  MENU_fileMenu,3,_enable, "/QQuit"
  'Beep menu ------
  MENU_beepMenu,0,_enable, "Beeper"
  MENU_beepMenu,1,_enable, "/1Beep once"
  MENU_beepMenu,2,_enable, "/2Beep twice"
END FN
```

These constants are defined to make the *doMenuEvnts* function easier to read and to add or remove menus later.

The *gWndNum* global variable holds the id of the next window to open. The *gProgramEnds* variable is used in the Main event loop to control when the program ends.

This function is called from the start of the program and whenever the "New Window..." menu item is selected from the File menu.

Each window is offset to the right and down six pixels so you can see windows behind it. The next line increments the global variable *gWndNum* to make sure the next window has a new id.

A simple push button centered in the window.

This function puts up two menus with two items each.

The hypen makes a dashed line menu item (used to keep menu items logically separated).

The back-slash character (/) indicates that the next letter is a command key equivalent.

continued next page...

```
LOCAL FN doDialogEvnts
  evnt = DIALOG(0)
  id   = DIALOG(evnt)
  SELECT evnt
    CASE _btnClick
      BEEP:BEEP:BEEP
    CASE _wndClick
      WINDOW id
    CASE _wndClose
      WINDOW CLOSE id
  END SELECT
END FN

LOCAL FN doMenuEvnts
  menuID = MENU (_menuID)
  itemID = MENU (_itemID)
  SELECT menuID
    CASE _fileMenu
      SELECT itemID
        CASE _fNewWnd
          title$="Window"+STR$(gWndNum)
          FN openWindow(title$,gWndNum)
        CASE _fQuit
          gProgramEnds = _true
      END SELECT
    CASE _BeepMenu
      SELECT itemID
        CASE _bBeep1
          BEEP
        CASE _bBeep2
          BEEP:BEEP
      END SELECT
  END SELECT
  MENU
END FN

'--------- Main program -------------
FN openWindow ("Window 1",1)
FN putUpMenus

ON DIALOG FN doDialogEvnts
ON MENU   FN doMenuEvnts

DO
  HANDLEEVENTS
UNTIL gProgramEnds
```

This function is called by HANDLEEVENTS when a DIALOG event occurs, as specified by ON DIALOG FN doDialogEvnts

Click the *beep3* button and you'll hear, you guessed it, three beeps.

Clicking a window brings it to the front.

Closes the window with the number "id" when you click that window's close box.

This function is called from HANDLEEVENTS when a menu event occurs, as defined with ON MENU FN doMenuEvnts.

These two statements get the menu item that was chosen. Command-key menu equivalents are also sent here.

Constants defined at the start of the program make this part of the program much easier to read. Imagine trying to figure out what was happening if numbers were used instead.

These two calls open the first window and put up the menus.

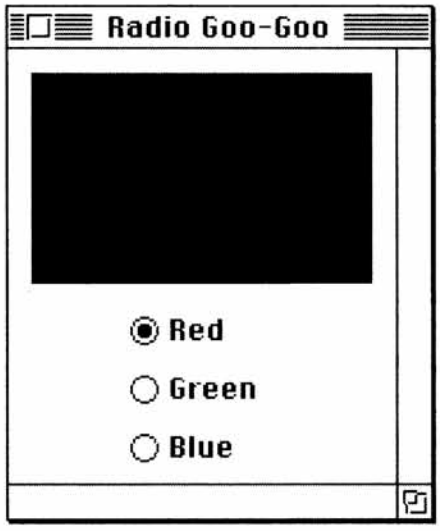
Defines which function HANDLEEVENTS should call when DIALOG events occur.

Defines which function HANDLEEVENTS should call when a MENU event occurs.

When an event occurs, HANDLEEVENTS calls the appropriate function defined with either ON DIALOG FN or ON MENU FN.

Exercise 19.3

Enter the program below, run it, and fill in what is missing in the window.




```
'--------- Constants --------
_rBtn1    = 1
_rBtn2    = 2
_rBtn3    = 3

'--------- Functions --------
LOCAL FN resetRadioGroup(btnClicked,F,L)
  FOR btnNum = F TO L
    LONG IF BUTTON(btnNum) = _markedBtn
      BUTTON #btnNum, _activeBtn
    END IF
  NEXT btnNum
  BUTTON btnClicked,_markedBtn
END FN

LOCAL FN colorize
  SELECT
    CASE BUTTON(_rBtn1) =_markedBtn
      COLOR _zRed
    CASE BUTTON(_rBtn2) =_markedBtn
      COLOR _zGreen
    CASE BUTTON(_rBtn3) =_markedBtn
      COLOR _zBlue
  END SELECT
  BOX FILL 10,10 TO 195,100
END FN

LOCAL FN doDialog
  evnt = DIALOG(0)
  id   = DIALOG(evnt)
  SELECT CASE evnt
    CASE _btnClick
      FN resetRadioGroup (id,_rBtn1,_rBtn3)
      FN colorize
  END SELECT
END FN

LOCAL FN initialize
  WINDOW 1, "Radio Goo-Goo",(0,0)-(220,200)
  BUTTON _rBtn1, _markedBtn, "Red",   (70,110)-(150,130),_radio
  BUTTON _rBtn2, _activeBtn, "Green",(70,135)-(150,155),_radio
  BUTTON _rBtn3, _activeBtn, "Blue", (70,160)-(150,180),_radio
  FN colorize
END FN
```

The *resetRadioGroup* function first makes sure all the radio buttons are unmarked, then marks the button that was clicked.

This function displays a color box depending on which radio button is marked.

When a radio button is clicked, a button clicked event occurs. This line passes the *resetRadioGroup* function the id of the radio button to mark, and the id's of the first and last radio buttons to reset

This line calls the function that figures out which button is marked and displays the appropriate color box.

continued next page...

```
'--------- Main ------------
FN initialize
ON DIALOG FN doDialog

DO
  HANDLEEVENTS
UNTIL cmdPeriod
```

Programming problems

Program problem 19.1

Read the *Buttons* chapter of the *Handbook.*

From *FutureBASIC,* open the example program in the *Buttons* chapter of the *Handbook* under *Example of push and shadow buttons.* All example programs in the *Handbook* come on disk in the *Examples* folder. They are always in the folder with the same chapter name, the *Buttons* folder in this case. The program is called *Push, Shadow.BAS.*

Loading example programs from disk is a lot easier than typing them in manually.

Modify the program so the shadow button is renamed "Cancel" and the push button is renamed "OK".

Program problem 19.2

From *FutureBASIC,* open the example program in the *Buttons* chapter of the *Handbook* under *Example of radio buttons.* The program is called *Radio.BAS.*

Modify the program so the radio buttons are renamed "Color", "Gray Scale", and "Black and white".

Program problem 19.3

Read the *Menus* chapter of the *Handbook.*

From *FutureBASIC,* open the example program in the *Menus* chapter of the *Handbook* under *Example of putting up a simple menu.* The program is called *Simple.BAS.*

Modify the program by adding an Apple menu called "Help..."

The examples in the Reference manual are in FutureBASIC Help (under the Apple menu). You can select and copy the examples from help windows and paste them into FutureBASIC and try them.

The Functions Library and Tech Notes Help also have example programs you can copy and paste into your programs. These functions are provided so you don't have to recreate the wheel every-time you need a common function.

Lesson 20

Using the mouse

The mouse is used for many things. In many cases, the use of the mouse generates an event. For example, if you click the close box of a window, a *_wndClose* event is generated. If you click a button, a *_btnClick* event is generated. Neither of these events will generate a mouse event, per se.

When the user clicks elsewhere, mouse events are generated. You will be able to determine:

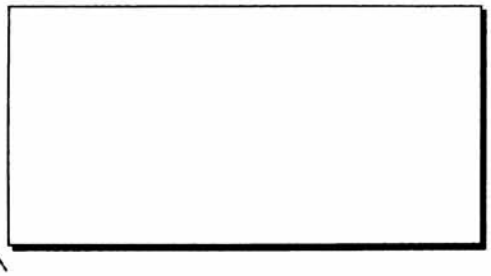
- if the mouse was clicked, double-clicked or triple-clicked
- if the mouse is in a click and drag event and if so, where the mouse was first clicked and where the mouse is currently.

Among other things, you will create a simple paint program that uses mouse positions to draw.

Exercise 20.1

Enter the program below, run it, and fill in what is missing in the window.

```
'-------- Functions -----------
LOCAL FN initializeWindow
  WINDOW 1,,(0,0)-(200,100),-_dialogShadow
  PRINT %(5,30) "Dbl-Click mouse in window to"
  PRINT %(5,45) "close this modal dialog box."
END FN

LOCAL FN doMouse
  SELECT MOUSE(0)      'type of mouse evnt
    CASE _click1, _click1nDrag
      LONG IF WINDOW(_activeWnd) = 1
        WINDOW CLOSE 1
        END
      END IF
  END SELECT
END FN

'-------- Main ---------------
FN initializeWindow
ON MOUSE FN doMouse

DO
  HANDLEEVENTS
UNTIL cmdPeriod
```



A window with a negative type is "modal". This means you cannot select menus or do other operations until the dialog window is closed. Modal means "a mode excluding all other operations or actions".

When the mouse is clicked, a *_click1* or a *_click1nDrag* event occurs. It's usually a *_click1nDrag* event because the mouse button is still down when it gets to the function.

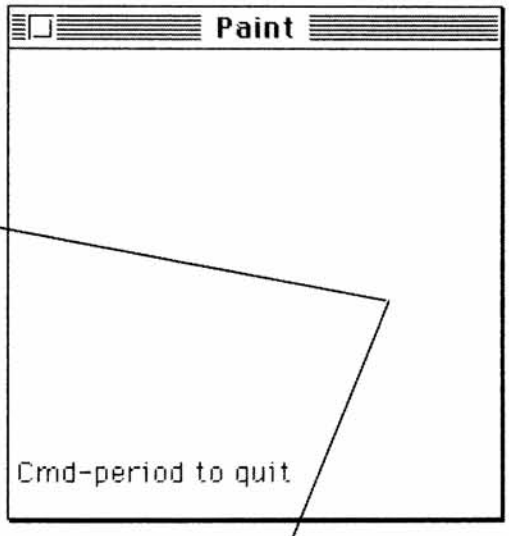
The dialog box is closed when the mouse is clicked in it.

This establishes which function HANDLEEVENTS will call when it encounters a mouse event.

Exercise 20.2

Enter the program below, run it, and fill in what is missing in the window. Save it when done.

```
RESOURCES "myFile.rsrc"

'--------- Constants ------------
_pencilCurs  = 134

'--------- Functions ------------
LOCAL FN initializeWindow
  WINDOW 1,"Paint",(0,0)-(210,300),_docNoGrow
  PRINT %(3,285)"Cmd-period to quit";
  CURSOR _pencilCurs
END FN

LOCAL FN moved (h,v)
  moved = _false
  IF MOUSE(_horz) <> h THEN moved =_true
  IF MOUSE(_vert) <> v THEN moved =_true
END FN = moved

LOCAL FN draw (h,v)
  WHILE MOUSE(0) = _click1nDrag
    LONG IF FN moved(h,v)
      h = MOUSE(_horz)
      v = MOUSE(_vert)
      PLOT TO h, v              'draw TO pos
    END IF
  WEND
END FN

LOCAL FN doMouse
  LONG IF MOUSE(0) = _click1nDrag
    h = MOUSE(_horz)
    v = MOUSE(_vert)
    PLOT h, v                   'pencil down
    FN draw (h,v)               'go draw
  END IF
END FN

'--------- Main -----------------
FN initializeWindow
ON MOUSE FN doMouse

DO
  HANDLEEVENTS
UNTIL cmdPeriod
```



This resource file should still be around. If not, create a new one as in *Exercise 18.4.*

This function returns true if the mouse has moved since last checked. This stops annoying cursor flash.

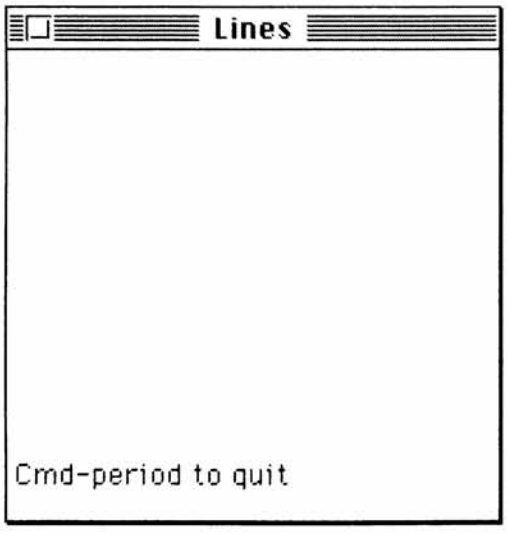
As long as the mouse is held down, this function draws from the last position to the current mouse position.

These mouse functions return the current mouse position. See *MOUSE function* in the *Reference* manual for more information.

This sets the starting point for drawing (where the mouse was first clicked).

Exercise 20.3

This is a slight modification of the previous exercise, so you won't need to retype everything. Modify *Exercise 20.2* so it looks like the program below, run it, and fill in what is missing in the window.




```
'--------- Functions ------------
LOCAL FN initializeWindow
  WINDOW 1,"Lines",(0,0)-(210,200),_docNoGrow
  PRINT %(3,185)"Cmd-period to quit";
  CURSOR _crossCursor
END FN

LOCAL FN moved (h,v)
  moved = _false
  IF MOUSE(_horz) <> h THEN moved =_true
  IF MOUSE(_vert) <> v THEN moved =_true
END FN = moved

LOCAL FN plotLine (h2,v2, h,v)
  PLOT h2,v2 TO h, v
  DELAY 100
  PLOT h2,v2 TO h, v
END FN

LOCAL FN drawLine (h2,v2)
  PEN ,,,_srcXOR
  WHILE MOUSE(0) = _click1nDrag
    LONG IF FN moved(h,v)
      h = MOUSE(_horz)
      v = MOUSE(_vert)
      FN plotLine (h2,v2, h,v)
    XELSE
      FN plotLine (h2,v2, h,v)
    END IF
  WEND
  CALL PENNORMAL
  PLOT h2,v2 TO h,v
END FN

LOCAL FN doMouse
  LONG IF MOUSE(0) = _click1nDrag
    h = MOUSE(_lastMHorz)    'get horz
    v = MOUSE(_lastMvert)    'get vert
    FN drawLine (h,v)        'draw line
  END IF
END FN
```

This function draws a line to the current mouse position, waits a tenth of a second, then draws the line again. Because *_srcXOR* mode is set, the second time the line is drawn it erases the first line, but without disturbing the background. See the *QuickDraw* section of the *Handbook* for more information about drawing modes.

When the mouse button is released, the pen is set back to normal and drawn one final time. The PENNORMAL function is a toolbox call described in the *QuickDraw* section of the *Handbook*.

continued next page...

continued from the last page...

```
'----------- Main -----------------
FN initializeWindow
ON MOUSE FN doMouse

DO
  HANDLEEVENTS
UNTIL cmdPeriod
```

Programming problems

Program problem 20.1

Read the two pages in the *Cursor resources* chapter of the *Handbook.*

From *FutureBASIC,* open the example program in the *Cursor resources* chapter of the *Handbook* under *Changing cursors.* All example programs in the *Handbook* come on disk in the *Examples* folder. The program is called *Cursor resources) Example.BAS.*

Modify the program so the black and white cursor (CURS) is a hand instead of a lasso.

The example program will not run unless the "ResFile.rsrc" file is in the same folder as the "Cursor Resources) Example.BAS" file. The resource file is in the Handbook folder. Make a copy of it and put it in the Cursor folder (be sure to take the "copy" part out of the name before running the program).

Program problem 20.2

Read the *Mouse* chapter of the *Handbook.*

From *FutureBASIC,* open the example program in the *Mouse* chapter of the *Handbook* under *Drawing with the mouse.* All example programs in the *Handbook* come on disk in the *Examples* folder. The program is called *Drawing.BAS.*

Modify the program so it beeps whenever the mouse is first clicked.

Lesson 21

More about windows

Windows are such an important part of the Macintosh interface, I thought a little more discussion would be in order.

This lesson will discuss several window issues, such as:

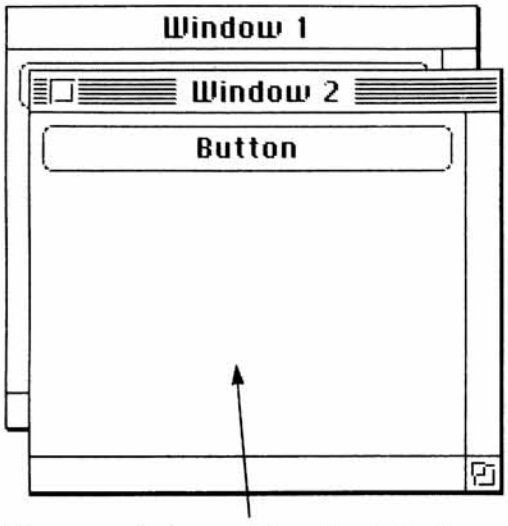
- how to direct output to a specific window, whether it's in front or in back.
- how to determine which window is on top.
- determine monitor/screen size so you can set window size.
- determine the width and height of windows. This is important with sizable windows.
- how and when to update the contents of a window.

This lesson will give you a feel for some of the window statements and functions available. They are by no means the only ones. For a more complete understanding of windows, read the Windows chapter of the Handbook.

Exercise 21.1

Refreshing windows correctly is important. Enter the program below, run it, and fill in what is missing in the windows.

```
'---------- Function ------------
LOCAL FN buildWindows
  WINDOW 1,"Window 1",(5,40)-(205,200)
  BUTTON 1,_enable,"Button",(5,5)-(180,25),_push
  WINDOW 2,"Window 2",(100,100)-(300,260)
  BUTTON 1,_enable,"Button",(5,5)-(180,25),_push
END FN

LOCAL FN doDialog
  evnt = DIALOG(0)
  id   = DIALOG(evnt)
  SELECT evnt
    CASE _wndClose
      WINDOW CLOSE id
    CASE _wndClick
      WINDOW id
    CASE _wndRefresh
      DELAY _sec
      PRINT%(3,45)"Window-"id"-updated!"
    CASE _btnClick
      where    = WINDOW(_activeWnd)
      IF where = 1 THEN BEEP
      IF where = 2 THEN BEEP:BEEP
  END SELECT
END FN

'---------- Main -------------
FN buildWindows
ON DIALOG FN doDialog
DO
  HANDLEEVENTS
UNTIL cmdPeriod
```



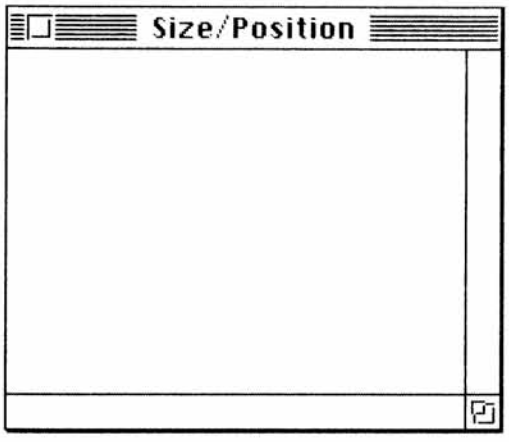
Move one window over the other, bring the back one to the front, and so on. Can you see when _wndRefresh events occur?

This window function returns the *id* of the top, active window. See the *WINDOW function* in the *Reference* manual for a description of this and other window functions.

Text and graphics you put in a window must be updated whenever a *_wndRefresh* event occurs. Buttons, edit fields, and menus all update themselves.

Exercise 21.2

A window's position and size can be determined using the WINDOW function. Enter the program below, run it, and fill in what is missing in the windows.




```
'---------- Function -----------
LOCAL FN buildWindows
  WINDOW 1,"Size/Position",(5,40)-(210,200)
  TEXT _geneva, 9
END FN

LOCAL FN printWindowData
  h    = ABS(WINDOW(_toLeft))
  v    = ABS(WINDOW(_toTop))
  wide = WINDOW(_width)
  high = WINDOW(_height)
  CLS
  PRINT %(13,20)"Window size:";
  PRINT wide "wide X";
  PRINT high "high"
  PRINT %(13,30) "Window position: ";
  PRINT "("h","v")"
  PRINT %(13,high-30)"Resize & move window."
  PRINT %(13,high-20)"Cmd-period to quit..."
  BOX 10,10 TO wide-10, high-10
END FN

LOCAL FN doDialog
  evnt = DIALOG(0)
  id   = DIALOG(evnt
  SELECT evnt
    CASE _wndClose
      WINDOW CLOSE id
    CASE _wndRefresh
      FN printWindowData
    CASE _preview
      LONG IF id = _wndMoved
        FN printWindowData
      END IF
  END SELECT
END FN

'---------- Main ----------------
FN buildWindows
ON DIALOG FN doDialog
DO
  HANDLEEVENTS
UNTIL cmdPeriod
```

This function returns the distance, in pixels, from the left side of the window to the edge of the screen.

This function returns the distance from the top of the window to the top of the screen.

These two functions return the width and height of the window.

If either a *_wndRefresh* or *_wndMoved* event occurs, the program updates and displays new window data.

Exercise 21.3

Information about the hardware being used, such as screen size, can be obtained with the SYSTEM function. Enter the program below, run it, and draw a sketch of how the window looks.

Screen width is calculated here to make a window 20 pixels from the edge of the monitor.

The SYSTEM(_cpuType) function returns a number from 1 to 5, corresponding to Motorola's chips 68000 through 68040. A simple calculation returns the cpu.

This function uses the SYSTEM function to get information about the computer the program is running on. See *SYSTEM function* in the *Reference* manual for more information.

```
'---------- Functions -------------
LOCAL FN makeWindows
  wide = SYSTEM(_scrnWidth)
  high = SYSTEM(_scrnHeight)
  WINDOW 1,"SysInfo",(20,20+36)-(wide-20,high-20)
  TEXT _monaco, 9
END FN

LOCAL FN cpu$(id)
  LONG IF id >=1 AND id <= 5
    type$ = RIGHT$(STR$((id-1)*10),2)
    cpu$ = " 680" + type$
  XELSE
    cpu$ = "id ="+STR$(id)+" (unknown)"
  END IF
END FN = cpu$

LOCAL FN printSystemData
  PRINT
  PRINT "Hardware information obtained"
  PRINT "with the "SYSTEM" function:"
  PRINT
  PRINT "Screen width   = "SYSTEM(_scrnWidth)
  PRINT "Screen height  = "SYSTEM(_scrnHeight)
  x& = SYSTEM(_memAvail)
  PRINT "Memory avail   = "USING"##,###,###";x&
  PRINT "Color hardware = "SYSTEM(_maxColors)"bits"
  PRINT "Current colors = "SYSTEM(_crntDepth)"bits"
  PRINT "CPU type       = "FN cpu$(SYSTEM(_cpuType))
  PRINT
  PRINT "cmd-period to quit"
END FN

'------------ Main --------------
FN makeWindows
FN printSystemData

DO
  HANDLEEVENTS
UNTIL cmdPeriod
```

Programming problems

Program problem 21.1

Read *Every window is an island* and the *Button and edit field id's are unique to a window* sections in the *Windows* chapter of the *Handbook*. It is important to understand that each window has its own font, size, style, pen width, pen position, color, pattern and so on.

Type in the program example on the *Every window is an island* page. Modify the pen width for window 2 to be 16,16.

Lesson 22

Sending output to a printer

Programs can send text and graphics to a printer the same way they send text and graphics to a window.

Instead of opening a window, use `ROUTE_toPrinter`, then draw all output normally. Just think of the printer paper as a window. All `PRINT` statements and graphic statements, like `PLOT` and `BOX`, will go to the printer. When you're finished printing, use `ROUTE_toScreen`.

Don't concern yourself too much with the type of printer connected. The printer driver for that printer will make output look like it would on the screen (better with text).

Important printing statements:

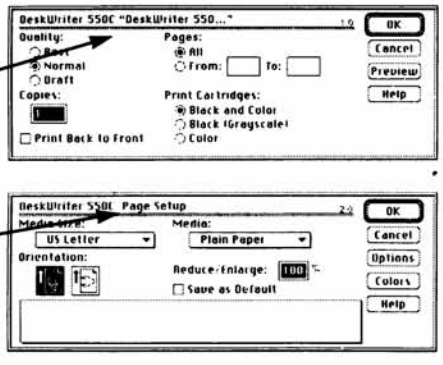
`ROUTE _toPrinter` Sends all output, including `PRINT`, `PLOT`, `BOX`, `PICTURE` and toolbox graphic routines, to the printer. Sort of like opening a window on the screen, except the printer paper is the window.

`ROUTE _toScreen` Sends output back to the screen. Think of it as the printer window being sent to the back and another window being brought to the front. The printer is still open and can be sent output by using `ROUTE _toPrinter` again.

`CLEAR LPRINT` Ejects the current page so you can start printing on a new piece of paper.

`CLOSE LPRINT` Signifies completion of the print job. The printer driver directs output to spooling routines if used, and frees up any memory being used by the printer driver.

`WINDOW(_width)` Returns the width of the printer paper.
`WINDOW(_height)` Returns the height of the printer paper. Use these two functions immediately after `ROUTE _toPrinter`, to determine page size and orientation.



`DEF LPRINT` Opens the standard *Print* dialog so the user can set printer attributes. This is usually invoked when *Print...* is chosen from the File menu.

`DEF PAGE` Opens the standard *Page Layout* dialog so the user can set print job attributes. This is usually invoked when *Page Setup...* is chosen from the File menu.

See the *Printer output* chapter of the Handbook for more information about printing.

Exercise 22.1

Enter the program below, run it, and fill in what is missing on the printed page.

```
'----------- Functions ------------
LOCAL FN buildWindow
  WINDOW 1, "Printing", (0,0)-(200,120)
END FN

LOCAL FN printStuff
  DEF LPRINT
  LONG IF PRCANCEL = _false
    ROUTE _toPrinter
      wide = WINDOW(_width)
      high = WINDOW(_height)
      PLOT 0,0 TO wide,high
      TEXT _helvetica, 24
      PRINT %(100,400) "Hello paper..."
    ROUTE _toScreen
    CLOSE LPRINT
  END IF

  TEXT _geneva, 12
  PRINT
  PRINT "Finished printing."
  PRINT "Cmd-period to quit."
END FN

'----------- Main -------------
FN buildWindow
FN printStuff

DO
  HANDLEEVENTS
UNTIL cmdPeriod
```

Brings up the *Print* dialog. Users expect this to happen before printing. If the *Cancel* button was pressed in the *Print* dialog, printing is aborted. The PRCANCEL function is used to detect this.

The size of the paper is retrieved and a line is drawn on the paper from corner to corner.

Text can be defined for the printer the same way it's defined for a window. Use PRINT% to position text anywhere on the paper.

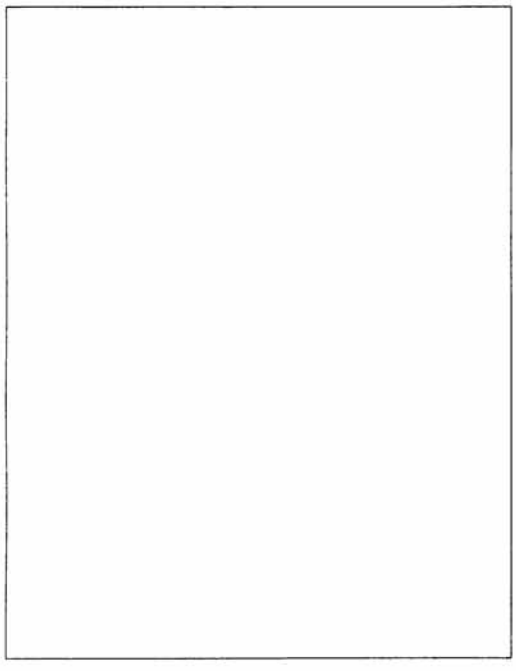
Always remember to close the print job. Otherwise printing will not be completed properly.

See PRHANDLE in the Reference manual for ways of determining settings in the Print and Page Setup dialogs.

Exercise 22.2

Setting coordinates for the printed page is sometimes much easier than using pixel coordinates. Enter the program below, run it, and fill in what is missing on the printed page.

```
'----------- Functions ------------
LOCAL FN buildWindow
  WINDOW 1, "Printing", (0,0)-(200,120)
END FN

LOCAL FN printCoordinates
  DEF LPRINT
  LONG IF PRCANCEL = _false
    ROUTE _toPrinter
    COORDINATE 850,1100
    PRINT %(10, 10)   "NW"
    PRINT %(830,10)   "NE"
    PRINT %(10, 1090)"SW";
    PRINT %(830,1090)"SE";
    PRINT %(415,565) "mid"
    BOX 100,100 TO 750,1000
  ROUTE _toScreen
  CLOSE LPRINT

  TEXT _geneva, 12
  PRINT
  PRINT "All done printing..."
  PRINT "Press cmd-period to quit."
END FN

'----------- Main ----------------
FN buildWindow
FN printCoordinates

DO
  HANDLEEVENTS
UNTIL cmdPeriod
```



This sets coordinates for the paper to be 850 x 1100, corresponding to a paper size of 8.5 by 11 inches. For example, this can be used to print a character 4.5 inches over by 5.5 inches down, using `PRINT%(450,550)`.

Graphics can be positioned the same way. This statement draws a box one inch from the edge of the page.

Most printers can't print to the edge of the paper so coordinates should reflect actual paper size obtained from PRHANDLE .

Programming problems

Program problem 22.1

Read the *Printer Output* chapter of the *Handbook.*

Modify *Exercise 18.3* to print the PICT resource to the printer instead of to the printer.

Lesson 23

Text edit fields

At the start of this book I showed you how to use INPUT to get characters typed from the keyboard.

As you probably figured out, INPUT is a poor method of getting keyboard input. It does not follow Apple's Macintosh interface guidelines because menus and other interface components are disabled while INPUT is in use. Nor can users use standard Macintosh techniques to edit text (such as cut, copy, and paste, click to place the insertion point, select text with the mouse, and so on).

Fortunately, *FutureBASIC* fully supports edit fields. The syntax for edit fields:

```
EDIT FIELD id, "title", rect, type, just
```

id	each edit field has its own id number.
title	text to appear in the edit field.
rect	the rectangle where the edit field appears.
type	edit fields can have borders or not, can be square or round, inverted or regular text, etc.
just	left, right, or center justify text.

This statement, without parameters, is also used to activate an edit field. For example, if there are 3 edit fields in a window and you want edit field 2 to be the one where keystrokes will go, use:

```
EDIT FIELD 2
```

To make all edit fields inactive (no key presses will go to an edit field), use:

```
EDIT FIELD 0
```

Edit fields refresh automatically when a *_wndRefresh* event occurs and can have scroll bars for viewing text. These, and other features make edit fields user friendly. Here's a sampling of statements and functions used with edit fields:

`EDIT$(id)` used to put a string into or get a string out of the edit field specified by id.

`TEKEY$ = s$` inserts string into the text at the current cursor position. If any text is selected it is replaced.

```
startSelect = WINDOW(_selStart)
```
returns the start of the selection in the current edit field.

```
endSelect = WINDOW(_selEnd)
```
returns the end of the selection in the current edit field. If *startSelect* = *endSelect* then no text is selected.

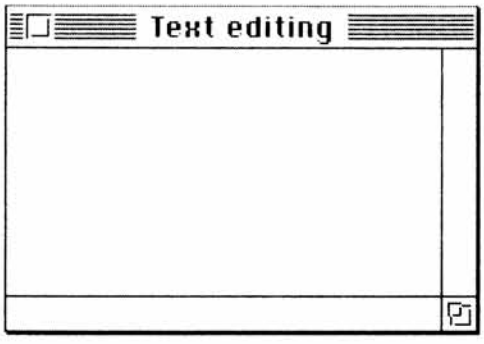
`SETSELECT` startPoint, endingPoint
used to select part or all of the text in the current edit field.

```
EDIT TEXT font, size, style, color
```
use this to change the font, size, style and color of selected text in the current edit field.

Exercise 23.1

Enter the program below, run it, and fill in what is missing in the window.

```
'----------- Functions ------------
LOCAL FN buildWindow
  WINDOW 1, "Text edit", (0,0)-(200,120)
  TEXT _geneva, 9
  EDIT FIELD 1,"ef1",(10,10)-(170,40),_framed
  EDIT FIELD 2,"ef2",(10,60)-(170,90),_framed
END FN

LOCAL FN putTextInEFs
  EDIT$(1) = "Edit fields are great!"
  EDIT$(2) = "Don't program without 'em"
end fn

'----------- Main -----------------
FN buildWindow
FN putTextInEFs

DO
  HANDLEEVENTS
UNTIL cmdPeriod
```



Edit fields are usually created in the same function that opens a window. Use `TEXT` to set the font, size and style for edit fields.

This function puts text in the edit fields. Getting text out of an edit field is the inverse of this. For example: `efText$ = EDIT$(1)` puts the contents of edit field 1 into the variable *efText$*.

Edit fields require a window and event handling in order to work.

Exercise 23.2

Enter the program below and run it.

```
'------------ Constants ----------------
_mFile   = 1
_iQuit   = 1
_mEdit   = 2
_iSelAll = 8

'------------ Functions ----------------
LOCAL FN initialize
  WINDOW 1,"Edit fields",(0,0)-(185,150)

  MENU _mFile,0,_enable,"File"'quit menu
  MENU _mFile,1,_enable,"Quit/Q"

  EDIT MENU _mEdit
  MENU _mEdit,7,_enable, "-"
  MENU _mEdit,8,_enable, "/ASelect All"

  TEXT  _geneva,9
  EDIT FIELD 1,"ef1",(10,10)-(160,40), _framedNoCR
  EDIT FIELD 2,"ef2",(10,50)-(160,80), _framedNoCR
  EDIT FIELD 3,"ef3",(10,90)-(160,120),_framedNoCR
END FN

LOCAL FN doMenus
  menuID = MENU(_menuID)
  itemID = MENU(_itemID)

  SELECT menuID
    CASE _mFile
      LONG IF itemID = _iQuit
        END
      END IF
    CASE _mEdit
      LONG IF itemID = _iSelAll
        SETSELECT 0,32767
      END IF
  END SELECT

  MENU
END FN
```

The `EDIT MENU` adds an Edit menu to your program that works with edit field text so text can be cut, copied and pasted.

This line selects all the text in the current edit field when *Select All* is chosen from the Edit menu. The number *32,767* is the maximum number of characters an edit field can hold.

Try typing text into the edit fields. Use Select All, Copy and Paste to transfer text to and from edit fields.

continued on the next page...

continued from previous page...

```
LOCAL FN doDialogs
  evnt = DIALOG(0)
  id   = DIALOG(evnt)
  crntEF = WINDOW(_EFnum)

  SELECT evnt
    CASE _EFreturn, _EFtab, _EFrightArrow, _EFdownArrow
      LONG IF crntEF = 3
        EDIT FIELD 1
      XELSE
        EDIT FIELD crntEF + 1
      END IF

    CASE _EFupArrow, _EFshiftTab, _EFleftArrow
      LONG IF crntEF = 1
        EDIT FIELD 3
      XELSE
        EDIT FIELD crntEF - 1
      END IF

    CASE _cursOver
      LONG IF id < 0
        CURSOR = _IBeamCursor
      XELSE
        CURSOR = _arrowCursor
      END IF
  END SELECT

END FN

'---------------- MAIN ----------------
FN initialize
ON MENU   FN doMenus
ON DIALOG FN doDialogs

DO
  HANDLEEVENTS
UNTIL quitMenuSelected
```

Gets the current edit field. This is used in several places below.

Checks to see if a <return>, <tab>, <right arrow> or <down arrow> key was pressed. If so, it activates the next edit field down, unless it's on the bottom edit field (3), in which case, it activates the top edit field (1).

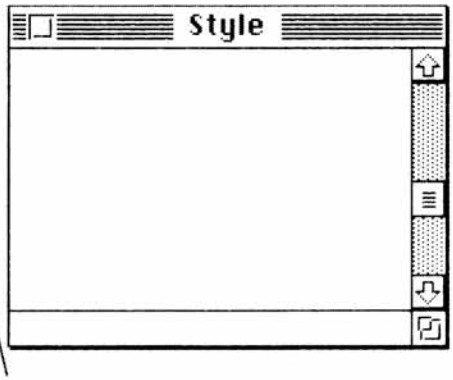
Checks to see if a <up arrow>, <shift-tab> or <left arrow> key was pressed. If so, it activates the next edit field above the current one, unless it's on the top edit field (1), in which case, it activates the bottom edit field (3).

When the cursor moves over a button or an edit field, a `_cursOver` event is generated. When the cursor moves over an edit field, the cursor needs to be changed to an *_IbeamCursor* for selecting text. If it's not over an edit field, it is changed to the standard arrow.

Move the cursor over edit fields to see the cursor change. Also press the keys mentioned above to see how control is moved from one edit field to another.

Exercise 23.3

Enter the program below, run it, and fill in what is missing in the window.




```
'---------- Functions --------------
LOCAL FN initialize
  WINDOW 1,"Style",(0,0)-(186,126)
  EDIT FIELD -1,,(0,0)-(171,111),_framed
  SCROLL BUTTON -1,1,1,1,1,,_scrollVert
END FN

LOCAL FN doDialogs
  evnt = DIALOG(0)
  id   = DIALOG(evnt)
  SELECT evnt
    CASE _preview
      LONG IF id = _wndSized
        wide = WINDOW(_width)
        wigh = WINDOW(_height)
        EDIT FIELD -1,,(0,0)-(wide-1,high-1)
      END IF
    CASE _cursOver
      LONG IF id < 0
        CURSOR = _IBeamCursor
      XELSE
        CURSOR = _arrowCursor
      END IF
  END SELECT
END FN

LOCAL FN fillEditFields
  EDIT$(1) = "This is not a ransom note!"
  FOR spce = 0 TO WINDOW(_EFtextLen)
    SETSELECT spce, spce+1
    red = RND(30000)+30000
    grn = 0
    blu = RND(30000)+10000
    sze = RND(14)+10
    EDIT TEXT _sysFont,sze,,,red,grn,blu
  NEXT
  EDIT TEXT _geneva,24,_boldBit%,,0,40000
  TEKEY$ = CHR$(13)
  TEKEY$ = CHR$(13)
  TEKEY$ = "I am not a kidnapper! Please"
  TEKEY$ = " excuse my wierd green example."
END FN
```

An EDIT FIELD with a negative id is a "stylized" edit field that allows multiple fonts, sizes, styles and colors in all or part of the text. This edit field uses the full window area.

A SCROLL BUTTON with the same negative id as a stylized edit field is *attached* to that edit field and supports scrolling text.

Since the edit field fills the entire window, its size must be adjusted when the window size is changed. This checks for a *_wndSized* event and fits the edit field to the window.

This puts text into the current edit field.

This loop counts through the characters in the edit field, selecting each character one at a time with SETSELECT, randomly changing size and color with EDIT TEXT. The function WINDOW(_EFtextLen) returns the total number of characters in the edit field.

Color is specified by using red, green and blue values. This allows you to specify any of 16.7 million colors. See LONG COLOR in the *Reference* manual for details.

TEKEY$ is used to insert text at the cursor position in the current edit field. If text is selected it is replaced. These lines add carriage returns to the edit field.

continued next page...

continued from the previous page...

```
'------------ Main -----------------
FN initialize
FN fillEditFields
ON DIALOG FN doDialogs

DO
  HANDLEEVENTS
UNTIL cmdPeriod
```

Programming problems

Program problem 23.1

Read the *Scroll Bar* chapter of the *Handbook.*

Open the example program named *Scroll Edit fields.BAS*. If you have a color computer, modify the program to make all the edit fields have red text.

Program problem 23.2

Read the *Text editing* chapter of the *Handbook.*

Open the example program named *Example1.BAS*.

Make the program run without beeping. Also make the "Phone:" edit field functional.

Parting notes...

I would love to say that you now know everything there is to learn about programming the Macintosh. Sorry. The learning process is just starting. Computer programming is a moving target. Just when you think you're getting the hang of it, they change the rules! Don't worry, *FutureBASIC* will be upgraded as the Macintosh evolves and we'll keep you informed about the changes.

Some resources you'll find useful in your *FutureBASIC* programming adventures:

Handbook

The handbook offers a tremendous amount of information about many subjects. Read over its table of contents to become familiar with the information available. Many of things you will want to do are illustrated in the program examples.

Reference manual

Remember that most of the *Reference* manual examples are in the *FutureBASIC* help system. Copy examples from the help window and paste them into your programs for testing.

Where to get answers to questions

Zedcor offers technical support to registerd owners of *FutureBASIC*. This can be invaluable if you get stuck. Call 602-795-3996 during regular business hours (you might get lucky and catch us there late sometimes). Technical-support is also available by modem. From *FutureBASIC*, choose *About FutureBASIC...*, under the Apple menu, for a list of Zedcor electronic addresses.

Where to go from here

Here's a list of some of the products available to help you use and learn *FutureBASIC:*

Inside BASIC magazine

For a monthly magazine full of tips and examples, as well as the latest information about programming the Macintosh, subscribe to *Inside BASIC* magazine. Contact Ariel Publishing, Box 398, Pateros, WA 98846, 509-923-2249.

A book that starts where this one leaves off...

Zedcor's own technical-support manager, Frank Turovich, has written a great book called *Learning FutureBASIC, Macintosh BASIC Power*. Order directly from Sentient Fruit, Box, 13362, Tucson, AZ 85732-3362. Cost of the book and accompanying diskette: $39.95 + s&h ($6 U.S., $8 Canada, $14 elsewhere). Send U.S. Check/Money order.

Database engine

If you're writing programs that need fast access to data, consider *B-Tree Helper* from *Maagreeable* software. They provide a set of functions for creating and manipulating data files. Call 612-559-1108.

Spreadsheet engine

If you're writing programs that could use spreadsheet capabilities, consider the *Spreadsheet Interface Module for FutureBASIC* from By Design, Inc. Call 608-831-5259.

Program interface maker

Read the *PG:PRO* brochure that came with *FutureBASIC*. This program allows you to draw program interfaces with simple graphic tools. It then writes the program interface for you in *FutureBASIC*. Some people like this method of creating interfaces. See the review in the July, 1993 issue of MacWorld magazine. Call Staz Software at 601-255-7085.

Inside Macintosh

When you're ready to write programs using toolbox functions and procedures, get *Inside Macintosh* from Apple Computer, Inc. and published by Addison Wesley. See the *Toolbox* section of the *Handbook* for a description of these books.

Think Reference 2.0

Unfortunately, *Inside Macintosh* books are expensive. If you want access to toolbox documentation on a small budget, consider Symantec's *Think Reference 2.0*, a cross-referenced database of *Inside Macintosh* toolbox functions and procedures. It allows you to find toolbox routines quickly, and explains them in easy to understand language. Available from most Macintosh mail-order firms for around $70.

Answers

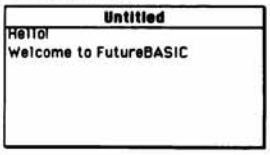
to exercises and programming problems

Lesson 0: Entering Programs

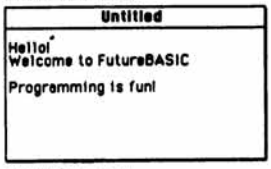
Just do the lesson.

Lesson 1: Printing to a window

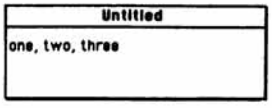
Exercise 1.1



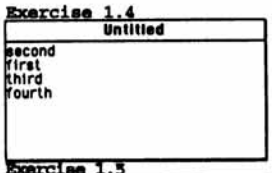
Exercise 1.2



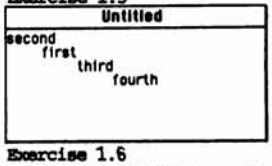
Exercise 1.3



Exercise 1.4



Exercise 1.5



Exercise 1.6

```
PRINT %(50,40) "Center of window";
```

Exercise 1.7

```
WINDOW 1,,(0,0)-(220,100)
PRINT  %(0,12)  "north west"
PRINT  %(140,12) "north east"
PRINT  %(0,85)  "south west";
```

Exercise 1.8

```
WINDOW 1,,(0,0)-(220,80)
PRINT %(85,20) "Andrew"
PRINT %(85,35) "Alex"
PRINT %(85,50) "Alexis";
PRINT %(85,65) "Andrea";
STOP
```

Programming problem 1.1

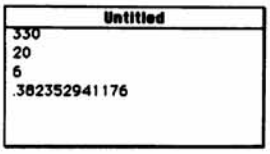
```
WINDOW 1,,(0,0)-(220,100)
PRINT  %(3,35) "Mike A. Gariepy"
PRINT  %(3,50) "4500 E. Speedway"
PRINT  %(3,65) "Tucson, AZ85712"
STOP
```

Programming problem 1.2

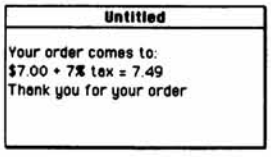
```
WINDOW 1,,(0,0)-(220,110)
PRINT  %(80,35) "Alabama"
PRINT  %(80,50) "Alaska"
PRINT  %(80,65) "Arizona"
PRINT  %(80,80) "Arkansas";
STOP
```

Lesson 2: Math operations

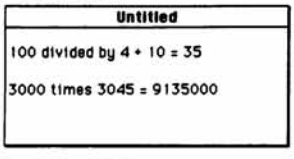
Exercise 2.1



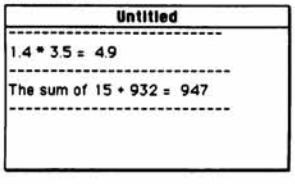
Exercise 2.2



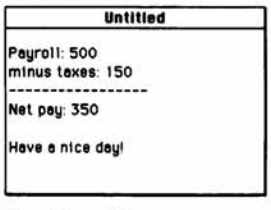
Exercise 2.3



Exercise 2.4



Exercise 2.5



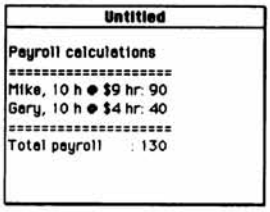
Exercise 2.6

```
PRINT "Multiply and divide are
done"
PRINT "BEFORE add and subtract,"
PRINT "sometimes returning un-
PRINT "expected results:"

PRINT "Use parentheses to force the"
PRINT "order of operations, so"
PRINT "answers are calculated as"
PRINT "you expect:"
```

Exercise 2.7

```
WINDOW 1,,  (0,0)-(210,130)
PRINT
PRINT "Answer with no parentheses:"
PRINT "15 + 20 * 5 ="15 + 20 * 5
PRINT "--------------------------"
PRINT "Answer with parentheses:"
PRINT "(15 + 20) * 5 ="(15 + 20) * 5
STOP
```

Exercise 2.8



Exercise 2.9

```
PRINT
PRINT "The correct answer to"
PRINT "10 * (10 + 10) is: "10 * (10 + 10);
STOP
```

Programming problem 2.1

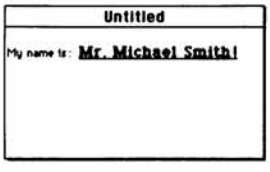
```
WINDOW 1,,  (0,0)-(210,150)
PRINT
PRINT "John,   35.5 h @ $9 hr:"; 35.5 * 9
PRINT "Mary,   26.9 h @ $9 hr:"; 26.9 * 9
PRINT "Larry,  22.3 h @ $9 hr:"; 22.3 * 9
PRINT "═══════════════════"
PRINT "Total payroll      :";(35.5+36.9+22.3)*9
STOP
```

Programming problem 2.2

```
WINDOW 1,,  (0,0)-(210,150)
PRINT
PRINT "Order of operations"
PRINT "500 + 200 * 23 ="(500 + 200) * 23
STOP
```

Lesson 3: Text Characteristics

Exercise 3.1



Exercise 3.2

```
TEXT  _newYork, 14, _boldBit%
```

Exercise 3.3

```
PRINT "company:     ABC Technical"
PRINT "address:      12 Main"
PRINT "    city:     Chicago
```

Exercise 3.4

```
PRINT "14 14 14"
TEXT ,18
```

Exercise 3.5

```
TEXT _newYork, 12, _normalText
PRINT %(2,45) "A long time ago, there was a"
PRINT "little girl named Goldilocks"
PRINT "who had long, blond hair."
PRINT
PRINT "One day she decided to go for"
PRINT "a walk in the forest. Before"
PRINT "long, she came upon a little"
PRINT "cottage with pretty flowers"
PRINT "all around it."
PRINT
PRINT ""I think I'll go in for a look.""
PRINT " she said to herself."
TEXT _geneva, 9
PRINT %(90,243) "page 1";
```

Exercise 3.6

```
PRINT %(3,20) "Lesson 3"

PRINT %(3,40) "Text characters"
TEXT _newYork, 12, _nil
PRINT %(3,60)"This lesson discusses the reasons"
PRINT %(3,75) "why we use text in the world.";
```

Programming problem 3.1

```
WINDOW 1,, (0,0)-(200,130)
PRINT
TEXT _geneva, 12, _boldBit% + _ulineBit%
PRINT "Years"
TEXT ,9,_nil
PRINT "1992"
PRINT "1993"
PRINT "1994"
PRINT "1995"
STOP
```

Programming problem 3.2

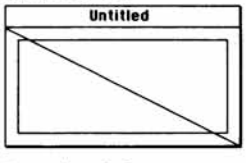
```
WINDOW 1,, (0,0)-(200,190)
TEXT _helvetica, 9, _nil
PRINT "Hello"
TEXT _helvetica, 18, _nil
PRINT "Hello"
TEXT _helvetica, 24, _nil
PRINT "Hello"
TEXT _helvetica, 36, _nil
PRINT "Hello"
STOP
```

Programming problem 3.3

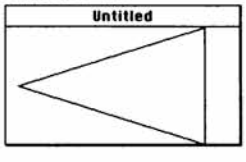
```
_normalText = 0
WINDOW #1,, (0,0)-(200,90)
TEXT _newYork,12,_boldBit%
PRINT %(3,20) "Monday"
PRINT %(3,35) "Tuesday"
PRINT %(3,50) "Wednesday"
TEXT,,_normalText
PRINT %(110,20) "Holiday"
PRINT %(110,35) "Work day"
PRINT %(110,50) "Work day"
STOP
```

Lesson 4: An introduction to graphics

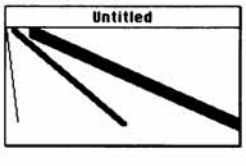
Exercise 4.1



Exercise 4.2



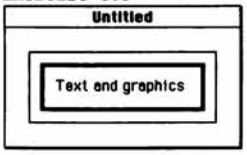
Exercise 4.3



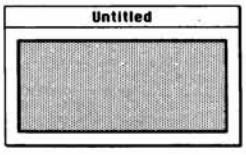
Exercise 4.4

```
PLOT 20,50 TO 170,50
PEN 8,8
```

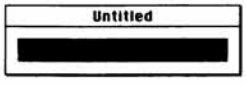
Exercise 4.5



Exercise 4.6



Exercise 4.7



Exercise 4.8

```
BOX FILL 10,10 TO 190,90
PEN 5,5,,,0
PLOT 0,0 TO 200,100
PLOT 200,0 TO 0,100
STOP
```

Exercise 4.9

Programming problem 4.1

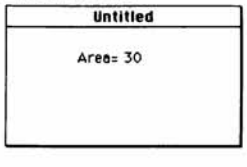
```
WINDOW 1,,(0,0)-(200,100), _docNoGrow
TEXT _geneva, 12, _boldBit%
PRINT %(60,45) "Graphics"
PEN 2,2
PLOT 20,50 TO 170,50
PLOT 20,55 TO 170, 55
STOP
```

Programming problem 4.2

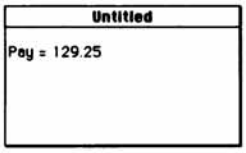
```
WINDOW 1,,(0,0)-(200,100), _docNoGrow
TEXT _geneva, 18, _nil
PRINT %(70,50) "HELLO"
PEN 2,2
PLOT 0,0 TO 200,100
PLOT 0,100 TO 200, 0
STOP
```

Lesson 5: Variable types for holding numbers

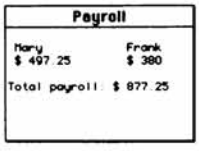
Exercise 5.1



Exercise 5.2



Exercise 5.3



Exercise 5.4

```
enrollment = toddlers + babys
```

Exercise 5.5

```
PRINT "Expense%: " rent% + electric% + water%
```

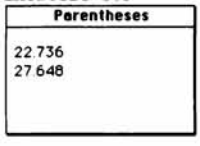
Exercise 5.6

```
second = 6789
PRINT "      " first
```

Exercise 5.7

```
BOX left%,top% TO right%,bottom%
Exercise 5.8
horz = 37
vert = 40
```

Exercise 5.9



Pogramming problem 5.1

```
WINDOW 1,"Payroll",(0,0)-(130,140)
tom!   = 123.34
larry! = 560.32
mary!  = 455.55
total! = tom! + larry! + mary!
PRINT: TEXT _monaco, 12
PRINT "ABC Corporation"
PRINT "Tom    "tom!
PRINT "Larry "larry!
PRINT "Mary   "mary!
PRINT "------------"
PRINT "Total=" total!
STOP
```

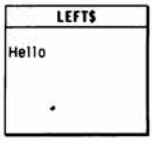
Programming problem 5.2

```
WINDOW 1,"Sums",(0,0)-(130,140)
first    = 22000
second   = 20000
sum% = first + second
sum! = first + second.
PRINT: TEXT _monaco, 12
PRINT "sum% ="sum%
PRINT "sum! ="sum!
STOP
```

Lesson 6: Text characters

Exercise 6.1

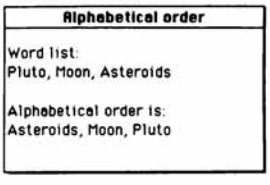
Exercise 6.2



Exercise 6.3

Exercise 6.4

Exercise 6.5



Exercise 6.6

```
lastName$  = "Henry"
state$     = "PA"
PRINT address$
```

Exercise 6.7

```
PRINT first$;first$;first$;first$;first$
```

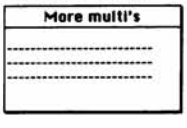
Exercise 6.8

```
lastName$  = RIGHT$(wholeName$,5)
PRINT "His first name is:
"firstName$
```

Exercise 6.9

```
PRINT STRING$(25,"c")
PRINT STRING$(25,".")
PRINT STRING$(20,"#");
STOP
```

Exercise 6.10



Programming problem 6.1

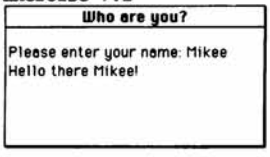
```
WINDOW 1,"Book Card",(0,0)-(260,110)
book$ = "For Whom the Bell Tolls"
author$ = "Earnest Hemingway"
TEXT _monaco, 12
PRINT
PRINT "Book name: " book$
PRINT "Author:    " author$
STOP
```

Programming problem 6.2

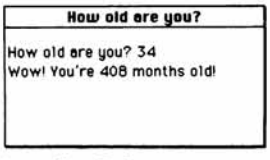
```
WINDOW 1,"Wedding Announcement",(0,0)-(280,130)
name1$ = "Frank Smith"
name2$ = "Mary Jones"
first1$ = LEFT$(name1$,5)
first2$ = LEFT$(name2$,4)
TEXT _newYork, 12
PRINT
PRINT "We are proud to announce the wedding"
PRINT "of "name1$" and "name2$
PRINT
PRINT first1$" and "first2$;
PRINT " will exchange vows."
STOP
```

Lesson 7: User input

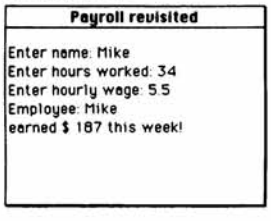
Exercise 7.1



Exercise 7.2



Exercise 7.3



Exercise 7.4

```
INPUT "Give me a number less than 100: "; num%
```

Programming problem 7.1

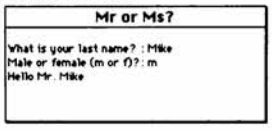
```
WINDOW #1,"Tallness",(0,0)-(200,100)
TEXT _monaco, 9
PRINT
INPUT "How tall are you in feet?: "; feet
INPUT "Plus how many inches?    : "; inches
PRINT
PRINT "You are ";feet*12 + inches"inches tall!"
STOP
```

Programming problem 7.2

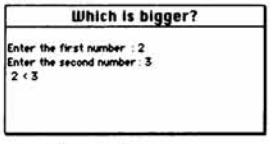
```
WINDOW #1,"Mail Call",(0,0)-(200,150)
TEXT _monaco, 9
PRINT
INPUT "Zip     : "; zip$
INPUT "Address: "; address$
INPUT "State   : "; state$
INPUT "City    : "; city$
Input "Name    : "; yourName$
PRINT
PRINT "Your address is:"
PRINT
PRINT yourName$
PRINT address$
PRINT city$", "state$" "zip$
STOP
```

Lesson 8: Comparing values

Exercise 8.1



Exercise 8.2



Exercise 8.3

```
INPUT "Is a baby lion a cub or a pup? ";ans$
  PRINT "That's correct!"

  PRINT "That's not correct!
```

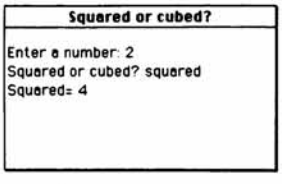
Exercise 8.4

```
PRINT "Guess my number (1-3)!"
INPUT "Your guess: ";ans

  PRINT "You lose!"
```

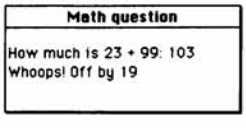
Exercise 8.5

```
XELSE

END IF
STOP
```

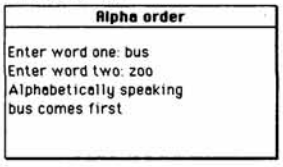
Exercise 8.6



Exercise 8.7



Exercise 8.8



Programming problem 8.1

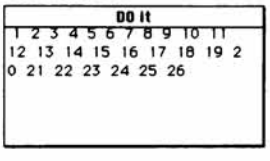
```
WINDOW #1,"Compare",(0,0)-(220,110)
PRINT
INPUT "Enter a number: ";num
LONG IF num = 100
  PRINT "= 100"
END IF
LONG IF num > 100
  PRINT "> 100"
END IF
LONG IF num < 100
  PRINT "< 100"
END IF
STOP
```

Programming problem 8.2

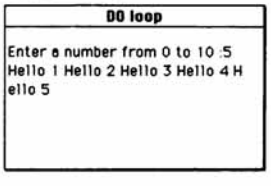
```
WINDOW 1,"Alpha Names",(0,0)-(230,110)
PRINT
INPUT "Enter name one: ";name1$
INPUT "Enter name two: ";name2$

PRINT "Alphabetically speaking...
LONG IF UCASE$(name1$) > UCASE$(name2$)
  PRINT name2$ " comes first"
XELSE
  PRINT name1$ " comes first"
END IF
STOP
```

Lesson 9: Repeating program lines (looping)

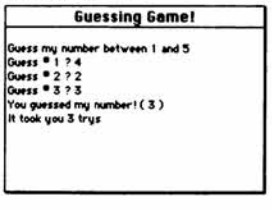
Exercise 9.1



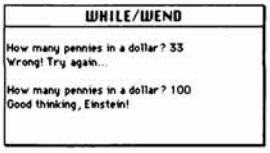
Exercise 9.2



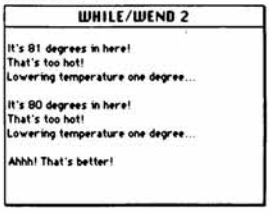
Exercise 9.3



Exercise 9.4



Exercise 9.5



Exercise 9.6

```
WINDOW #1,"WHILE/WEND 2",(0,0)-(220,150)
TEXT _geneva, 9
counter = 20
WHILE counter > 0
  PRINT counter;
  counter = counter - 1
  TRON x
WEND
STOP
```

Exercise 9.7

```
WINDOW #1,"ENDLESS LOOP 2",(0,0)-(220,150)
TEXT _geneva, 9
counter = 20
DO
  PRINT counter;
  counter = counter + 1
  TRON x
UNTIL counter > 100
STOP
```

Exercise 9.8

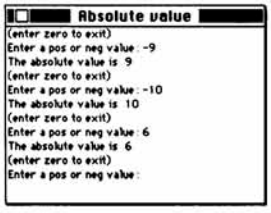
The program waits for a key to be pressed or the mouse button to be clicked before quitting. It replaces the STOP method of quitting (there is no longer a stop window opened to interfere with the window contents).

Programming problem 9.1

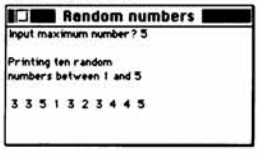
```
WINDOW #1,"2 + 2",(0,0)-(220,150)
TEXT _geneva, 9
trys = 0       'initialize counter
DO
  trys = trys + 1  'how may trys?
  INPUT "What is 2+2? ";answer
  LONG IF answer <> 4
    PRINT "You made a mistake!"
  END IF
UNTIL answer = 4
PRINT "Very good! It took you"trys"trys!"
STOP
```

Lesson 10: Some useful numeric functions

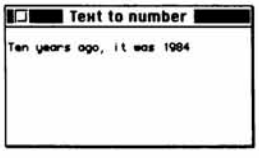
Exercise 10.1


Absolute value
(enter zero to exit)
Enter a pos or neg value: -9
The absolute value is 9
(enter zero to exit)
Enter a pos or neg value: -10
The absolute value is 10
(enter zero to exit)
Enter a pos or neg value: 6
The absolute value is 6
(enter zero to exit)
Enter a pos or neg value:

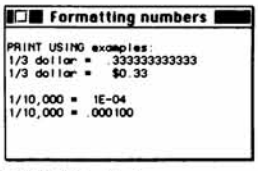
Exercise 10.2



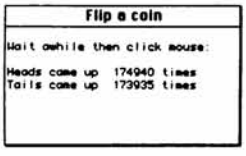
Exercise 10.3



Exercise 10.4



Exercise 10.5



Exercise 10.6

```
PRINT "Dick   " USING"$##,###.##";dick! * dMax!
PRINT "Mary   " USING"$##,###.##";mary! * mMax!
```

Exercise 10.7

```
PRINT "Year is: "VAL(RIGHT$(tDate$,2)) + 1900
```

Exercise 10.8

```
PRINT
PRINT "Total hours: "VAL(hrs1$) + VAL(hrs2$)

DO
UNTIL LEN(INKEY$) OR FN BUTTON
```

Exercise 10.9

```
num2 = RND(100)       'get random number 2

PRINT "     " num1

PRINT "Questions : ",total
PRINT "Correct   : ",correct
PRINT "Wrong     : ",wrong
```

Programming problem 10.1
Convert the three lines below:
WINDOW #1,"Subtraction test",(0,0)-(220,140)

```
PRINT "    -" num2

LONG IF answer = num1 - num2
```

Programming problem 10.2
Change the three lines below and convert all occurrances of the variable "answer" to "answer!" in case answers get bigger than short integers limit of 32,767.

```
WINDOW #1,"Multiplication test",(0,0)-(220,140)

PRINT "   x" num2

LONG IF answer! = num1 * num2
```

Programming problem 10.3

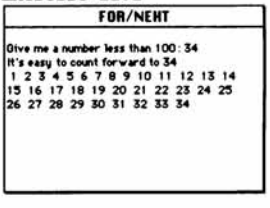
```
'------ Global variables -------
DIM array(200000)
END GLOBALS

'------ Functions --------------
LOCAL FN initializeWindow
  WINDOW 1,"Sorting",(0,0)-
(300,280),_docNoGrow
  TEXT _monaco, 9
END FN

'------- Main program ----------
FN initializeWindow

DO
  PLOT
RND(WINDOW(_width)),RND(WINDOW(_height))
UNTIL FN BUTTON
```

Programming problem 10.4

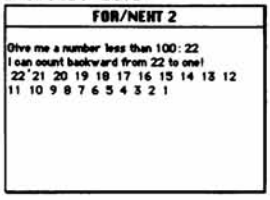
```
'------- Functions ----------
LOCAL
DIM myRGB.rgbColor
LOCAL FN drawColorLines
  blue%  = RND(256)*256
  green% = RND(256)*256
  red%   = RND(256)*256
  LONG COLOR blue, green,red
  PLOT TO RND(WINDOW(_width)),RND(WINDOW(_height))
END FN

LOCAL FN initialize
  WINDOW #1, "ColorLines",,_doc
END FN

'-------- Main ---------------
FN initialize
DO
  FN drawColorLines
UNTIL LEN(INKEY$) OR FN BUTTON
```

Lesson 11: Counter loops

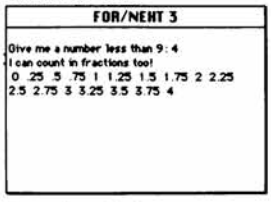
Exercise 11.1



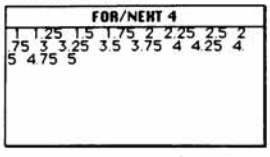
Exercise 11.2



Exercise 11.3

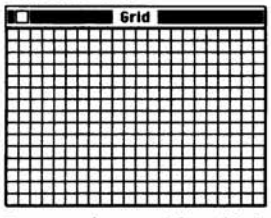


Exercise 11.4



Note: Change the variable count to count!

Exercise 11.5



Programming problem 11.1

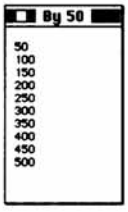
```
WINDOW #1,"Gross sales by
year",(0,0)-(240,220)
TEXT _monaco, 9
PRINT

FOR year = 1980 TO 1995
  PRINT year,
  PRINT USING "$##,###,###";RND(30000)*1000!
NEXT

DO
UNTIL FN BUTTON OR LEN(INKEY$)
```

Programming problem 11.2

```
WINDOW #1,"1-7/33",(0,0)-
(140,120)
TEXT _monaco, 9
PRINT

FOR num = 1 TO 7
  PRINT num ;
  PRINT USING ".##############";num/33
NEXT

DO
UNTIL FN BUTTON OR LEN(INKEY$)
```

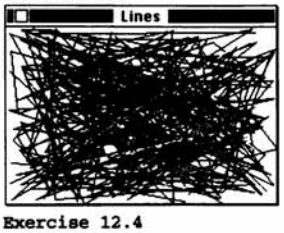
Lesson 12: Functions

Exercise 12.1



Exercise 12.2

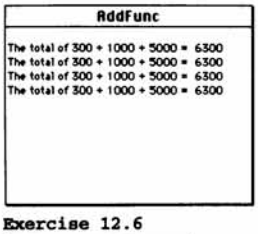
Exercise 12.3



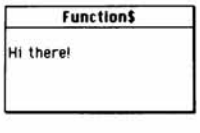
Exercise 12.4

```
'--------- Functions --------
LOCAL FN initializeWindow
  WINDOW 1,"By -50",(0,0)-(120,150)
  TEXT _monaco, 9
  PRINT
END FN

LOCAL FN doForLoop
  FOR x = 500 TO 50 STEP -50
    PRINT x
  NEXT
END FN

'--------- Main program ---------
FN initializeWindow
FN doForLoop

DO
UNTIL FN BUTTON OR LEN(INKEY$)
```

Exercise 12.5



Exercise 12.6



Programming problem 12.1

```
'-------- Sub-programs --------
LOCAL FN initializeWindow
  WINDOW #1,"Gross sales by
year",(0,0)-(240,220)
  TEXT _monaco, 9
  PRINT
END FN

LOCAL FN printData
  RANDOMIZE  'seed random number
  FOR year = 1980 TO 1995
   PRINT year ;
   PRINT USING"$##,###,###";RND(30000)*1000!
  NEXT
END FN

'--------- Main program ---------
FN initializeWindow
FN printData

DO
UNTIL FN BUTTON OR LEN(INKEY$)
```

Programming problem 12.2

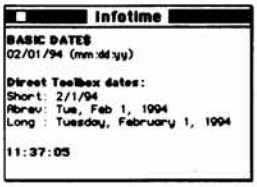
```
'--------- Sub-programs ----------
LOCAL FN initializeWindow
  WINDOW #1,"Payroll Revisited",(0,0)-(220,100)
  TEXT _monaco, 9
END FN

LOCAL FN doPayrollTaxes
  tom! = 23833.33   :tTax! = .23
  dick! = 32323.32  :dTax! = .27
  mary! = 78232.23  :mTax! = .38

  PRINT: PRINT "Employee Annual Taxes"
  PRINT "----------------------"
  PRINT "Tom  "USING"$##,###.##";tom! *tTax!
  PRINT "Dick "USING"$##,###.##";dick!*dTax!
  PRINT "Mary "USING"$##,###.##";mary!*mTax!
END FN

'------- Main Program ---------
FN initializeWindow
FN doPayrollTaxes

DO
UNTIL LEN(INKEY$) OR FN BUTTON
```

Programming problem 12.3

```
'--------- Functions --------
LOCAL FN initializeWindow
  WINDOW 1,"Average",(0,0)-(160,120)
  TEXT _geneva, 12
END FN

LOCAL FN average (a, b, c)
  d! =  (a + b + b)/3
END FN = d!

LOCAL FN doExample
  avg! = FN average (99,231,322)
  PRINT
  PRINT "The average of 99, 231"
  PRINT "and 322 is:"; avg!
END FN

'--------- Main program ---------
FN initializeWindow
FN doExample

STOP
```

Lesson 13: Text functions

Exercise 13.1



Exercise 13.2

Exercise 13.3

Length
The ASCII code of the first character, "m", is: 109
There are 4 characters in the word: "mike"
Uppercase word: MIKE
Press <return> to quit
Enter a word:

Exercise 13.4

Simple keys
(click mouse to quit)
This is a simple keyboard entry thing that is not as good as edi t fields. It's just an example

Programming problem 13.1

```
'--------- Functions --------
LOCAL FN initializeWindow
  WINDOW #1,"animate",(0,0)-(225,170)
END FN
LOCAL FN scanKeys
  key$ = INKEY$
  LONG IF LEN(key$)
    CLS
    FOR pointsize = 4 TO 64 STEP 2
      TEXT _geneva,pointsize,,_srcCopy
      PRINT %(75,80);key$;
    NEXT
  END IF
END FN
'--------- Main  ---------
FN initializeWindow
DO
  FN scanKeys
UNTIL FN BUTTON
```

Programming problem 13.2

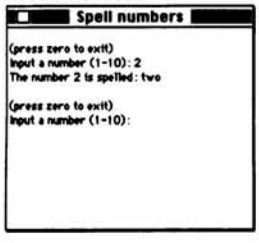
```
'--------- Functions --------
LOCAL FN initializeWindow
  WINDOW 1,"Lease",(0,0)-(225,170),_docNoGrow
  TEXT _monaco, 9       'window font
  PRINT
END FN

'--------- Main program ---------
FN initializeWindow

DO
  new$ = ""             'clear it
  INPUT "enter a string: ";st$
  WHILE LEN(st$) > 0
    char = ASC(st$)
    st$  = RIGHT$(st$,LEN(st$)-1)
    LONG IF char >= ASC("A") AND char <=
ASC("Z")
      char = char + 32
    END IF
    new$ = new$ + CHR$(char)
  WEND
  PRINT "Lowercase: "new$
  PRINT
UNTIL FN BUTTON OR LEN(new$) = 0
```

Lesson 14: Reading data in memory

Exercise 14.1

Exercise 14.2

Exercise 14.3

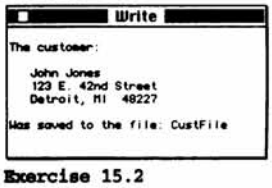


Programming problem 14.1

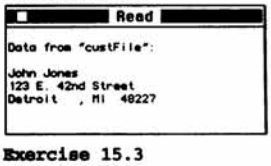
```
'--------- Data statements ----
DATA "January", "February", "March"
DATA "April"  , "May"     , "June"
DATA "July"   , "August"  , "September"
DATA "October", "November", "December"

'--------- Sub-programs --------
LOCAL FN initializeWindow
  WINDOW #1,"Months",(0,0)-(115,170)
  TEXT _monaco, 9              'window font
END FN

LOCAL FN spellNumbers          'list all data
  PRINT
  FOR count = 1 TO 12
    READ month$                'read data item
    PRINT USING "##";count;
    PRINT ": " month$
  NEXT
END FN

'--------- Main program ---------
FN initializeWindow            'set up window
FN spellNumbers
DO                             'wait to exit
UNTIL FN BUTTON OR LEN(INKEY$)
```

Lesson 15: Files

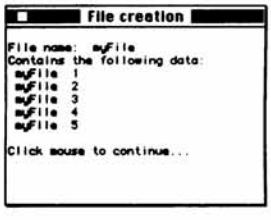
Exercise 15.1



Exercise 15.2



Exercise 15.3



Programming problem 15.1

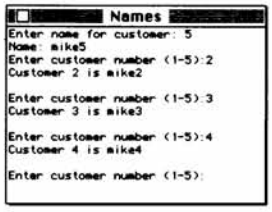
```
'--------- Sub-programs --------
LOCAL FN initializeWindow
  WINDOW 1,"Write",(0,0)-(220,110)
  TEXT _monaco, 9
END FN

LOCAL FN saveFileAs$
  filename$=FILES$(_fSave,"Save as...","",vol%)
  LONG IF LEN(filename$)=0
    filename$ = "cancelled"
  XELSE
    OPEN "O",1,filename$,,vol%
  END IF
END FN = filename$

LOCAL FN writeData             'write to disk
  filename$ = FN saveFileAs$
  LONG IF filename$ <> "cancelled"
    FOR x = 1 TO 100
      rndNum% = RND(32000)
      WRITE #1, rndNum%
    NEXT
    CLOSE #1                   'close the file
    PRINT "All done writing file: "filename$
    PRINT "Key or click mouse to quit..."
  END IF
END FN

'--------- Main program ---------
FN initializeWindow            'set up window
FN writeData                   'write data to disk

DO                             'wait to exit
UNTIL FN BUTTON OR LEN(INKEY$)
```

Programming problem 15.2

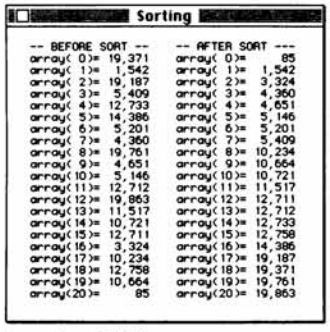
```
'--------- Sub-programs --------
LOCAL FN initializeWindow
  WINDOW 1,"Read",(0,0)-(220,200)
  TEXT _monaco, 9
END FN

LOCAL FN openFile$
  filename$ = FILES$(_fOpen,,,vol%)
  LONG IF LEN(filename$)=0
    filename$ = "cancelled"
  XELSE
    OPEN "I",1,filename$,,vol%
  END IF
END FN = filename$

LOCAL FN readData              'read from disk
  filename$ = FN openFile$
  LONG IF filename$ >< "cancelled"
    FOR x = 1 TO 100
      READ #1, rndNum%
      PRINT x"="USING "##,###";rndNum%
    NEXT
    CLOSE #1                   'close the file
    PRINT
    PRINT "All done reading: "filename$
    PRINT "Key or click mouse to quit..."
  END IF
END FN

'--------- Main program ---------
FN initializeWindow            'set up window
FN readData                    'write data to
disk

DO                             'wait to exit
UNTIL FN BUTTON OR LEN(INKEY$)
```

Lesson 16: Array Variables

Exercise 16.1



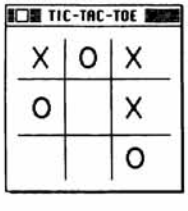
Exercise 16.2



Exercise 16.3

Exercise 16.4

Exercise 16.5



Programming problem 16.1

```
'--------- Globals ----------
DIM month$(12)
END GLOBALS

'--------- Data -------------
DATA January, February, March
DATA April, May, June, July
DATA August, September, October
DATA November, December

'--------- Functions --------
LOCAL FN initializeWindow
  WINDOW #1,"Months",(0,0)-(150,160)
  TEXT _monaco, 9
END FN

LOCAL FN loadArray
  FOR x = 1 TO 12
    READ month$(x)
  NEXT
END FN

LOCAL FN printArray
  CLS
  PRINT
  PRINT "The 12 monthes are:"
  FOR x = 1 TO 12
    PRINT month$(x)
  NEXT
END FN

'--------- Main program ---------
FN initializeWindow
FN loadArray
FN printArray

DO
UNTIL FN BUTTON OR LEN(INKEY$)
```

Programming problem 16.2

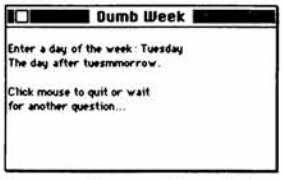
```
'------- Global variables --------
DIM array(2000)
END GLOBALS

'------- Functions ----------
LOCAL FN initializeWindow
  WINDOW 1,"Sorting",(0,0)-(300,280)
  TEXT _monaco, 9
END FN

LOCAL FN printArray (horzPos, message$)
  PRINT %(horzPos,20);message$
  FOR count = 1980 TO 2000
    PRINT %(horzPos,((count-1980)+3)*10);
    PRINT "array(";
    PRINT USING "####";count;")= ";
    PRINT USING "##,###";array(count)
  NEXT
END FN

LOCAL FN fillArrayWithRandomNums
  FOR count = 0 TO 2000
    array(count) = RND(20000)
  NEXT
  FN printArray(20,"-- BEFORE SORT --")
END FN

LOCAL FN sortArray
  start$ = TIME$
  FOR current = 0 TO 2000
    FOR look = current TO 2000
     LONG IF array(current) >= array(look)
        SWAP array(current), array(look)
     END IF
    NEXT
  NEXT
  FN printArray(145,"— AFTER SORT ---")
PRINT
PRINT "   start/end = "start$" - "TIME$;
END FN

'----- Main program --------
FN initializeWindow
FN fillArrayWithRandomNums
FN sortArray

DO
UNTIL FN BUTTON
```

Lesson 17: Selecting one case out of many

Exercise 17.1

Exercise 17.2

Exercise 17.3

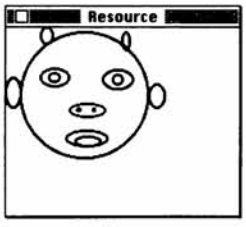


Programming problem 17.1

```
'--------- Constants -----------
_printHello   = 1      'Define a list
_beepOnce     = 2      'of commands for
_beepTwice    = 3      'the program
_printGoodbye = 4      'to do
_beepThrice   = 5
_haveANice    = 6

'--------- Functions --------
LOCAL FN initializeWindow
  WINDOW 1,"Do command",(0,0)-(150,160)
  TEXT _geneva, 9,,_srcCopy
  PRINT
  PRINT "<1> Print Hello"
  PRINT "<2> Beep once"
  PRINT "<3> Beep twice"
  PRINT "<4> Say goodbye and quit"
  PRINT "<5> Beep three times
  PRINT "<6> Have a nice day!"
END FN

LOCAL FN doCommand
  quit = _false
  what = VAL(INKEY$)
  SELECT what
    CASE _printHello
      PRINT %(5,120);: CLS LINE
      PRINT "Hello!";
    CASE _beepOnce
      BEEP
    CASE _beepTwice
      BEEP:BEEP
    CASE _beepThrice
      BEEP:BEEP:BEEP
    CASE _printGoodbye
      PRINT %(5,120);: CLS LINE
      PRINT "Goodbye!"
      quit = _true
      DELAY 1000
    CASE _haveANice
      PRINT %(5,120);: CLS LINE
      PRINT "Have nice day!";
  END SELECT
END FN = quit

'--------- Main program ---------
FN initializeWindow

DO
  quit = FN doCommand
UNTIL FN BUTTON OR quit = _true
```

Lesson 18: ResEdit and Resources

Exercise 18.3



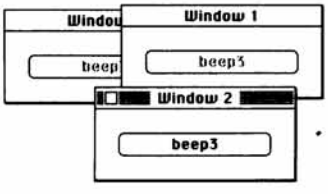
Exercise 18.5

A pencil cursor appears when you hold in the mouse. That's all.

Lesson 19: Events, buttons and menus

Exercise 19.1

Exercise 19.2



Exercise 19.3

Programming problem 19.1
Reverse the quoted strings in the button statements in the initialize function.

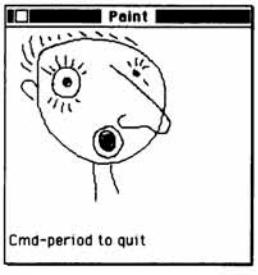
Programming problem 19.2
Change the quoted strings in the button statements in the initialize function.

Programming problem 19.3
Add APPLE MENU "Help..." to initialize function.

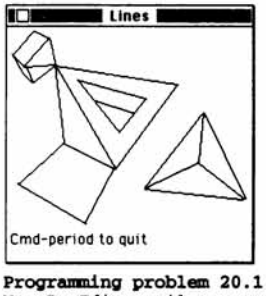
Lesson 20: Using the mouse

Exercise 20.1
The window closes when you click the mouse.

Exercise 20.2



Exercise 20.3



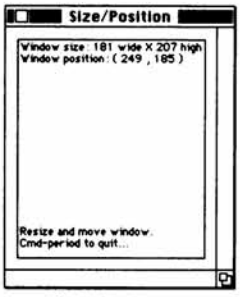
Programming problem 20.1
Use ResEdit until you get it.

Programming problem 20.2
Put a beep as the first line of the mousercise function.

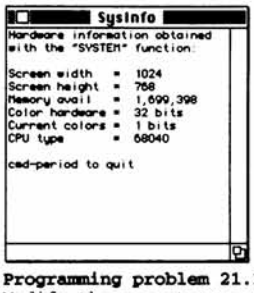
Lesson 21: More about windows

Exercise 21.1
Move the windows around. Study the program to see how it works.

Exercise 21.2



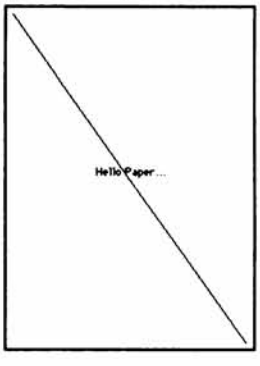
Exercise 21.3



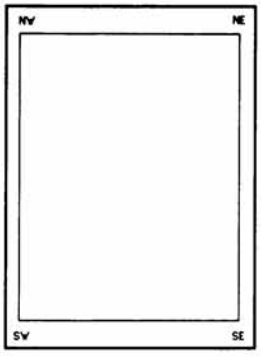
Programming problem 21.1
Modify the pen statement for window 2 to: PEN 16,16

Lesson 22: Sending ouput to a printer

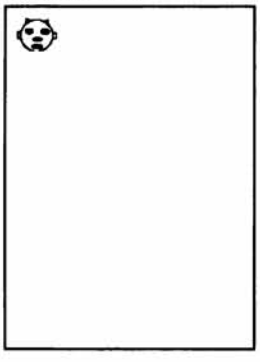
Exercise 22.1



Exercise 22.2

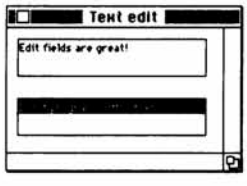


Exercise 22.3

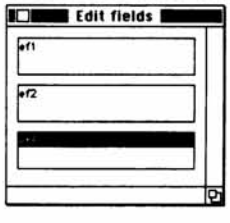


Lesson 23: Text edit fields

Exercise 23.1



Exercise 23.2
Be sure to try the Tab, arrow, return, and shift-tab keys.



Exercise 23.3

```
Congratulations!!!! Happy program-
ming!
```

Zedcor, Inc. • 4500 E. Speedway Suite 22 • Tucson, AZ 85712-5305

PROGRAMMING THE MACINTOSH® WITH FUTUREBASIC™